A METAMODEL OF OPERATIONAL CONTROL FOR DISCRETE EVENT LOGISTICS SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Timothy A. Sprock

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial & Systems Engineering

Georgia Institute of Technology
May 2016

A METAMODEL OF OPERATIONAL CONTROL FOR DISCRETE EVENT LOGISTICS SYSTEMS

Approved by:

Dr. Leon F. McGinnis, Advisor
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Christos Alexopoulos
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Nagi Gebraeel
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Benoit Montreuil
School of Industrial & Systems Engineering
Georgia Institute of Technology

Dr. Edward Huang
Systems Engineering & Operations Research
George Mason University

Date Approved: December 1, 2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AI artificial intelligence.

ALPS A Language for Process Specification.

AMHS automated material handling system.

AMS automated manufacturing systems.

ANN artificial neural networks.

AS/RS automated storage and retrieval system.

BOM bill of material.

CAD computer aided design.

CAMX Computer Aided Manufacturing using XML.

CMSD Core Manufacturing Simulation Data.

CPFR collaborative planning, forecasting, and replenishment.

CVRP capacitated vehicle routing problem.

CVRPPD CVRP with pick-ups and drop-offs.

DCS distributed control systems.

DELS Discrete Event Logistics Systems.

DES discrete event simulation.

DSL domain specific language.

ECA event-condition-action [rules].

ERP enterprise resource planning.

FCFS first come, first served.

FJSP flexible job-shop scheduling problem.

FJSP-PPF flexible job-shop scheduling problem.

FSM finite state machine.

ISA-95/L4 ISA-95 Level 4.

LIFO last-in, first-out.

MAUT multi-attribute utility theory.

MDP Markov decision process.

MES manufacturing execution systems.

MHS material handling system.

MPSG message part-state graph.

MRO maintenance, repair, and overhaul.

MRP material requirements planning.

OCL Object Constraint Language.

OMG Object Management Group.

OOP object-oriented programming.

PDM product data management.

PLC programmable logic controller.

PLM product lifecycle management.

PPRF product, process, resource, and facility.

PRS production-rule system.

PSL Process Specification Language.

SCADA supervisory control and data acquisition.

SCOR supply chain operation reference.

SDVRP split delivery vehicle routing problem.

SKU stock keeping unit.

SOA service-oriented architecture.

TFN token flow network.

TMS transportation management system.

UML Unified Modeling Language.

VMI vendor managed inventory.

VRP vehicle routing problem.

WCS warehouse control system.

WIP work in process.

WMS warehouse management system.

XML Extensible Markup Language.

SUMMARY

Discrete Event Logistics Systems (DELS) are a class of dynamic systems that are defined by the transformation of discrete flows through a network of interconnected subsystems. The DELS domain includes systems such as supply chains, manufacturing systems, transportation networks, warehouses, and health care delivery systems. Advancements in computer integrated manufacturing and intelligent devices have spurred a revolution in manufacturing. These smart manufacturing systems utilize technical interoperability and plant-wide integration at the device-level to drive production agility and efficiency. Extending these successes to enterprise-wide integration and decision-making will require the definitions of *control* and *device* to be extended and supported at the operations management and the business planning levels as well. In the future, smart operational control mechanisms must not only integrate real-time data from system operations, but also formulate and solve a wide variety of optimization analyses quickly and efficiently and then translate the results into executable commands.

However in contemporary DELS practice, these optimization analyses, and analyses in general, are often purpose-built to answer specific questions, with an implicit system model and many possible analysis implementations depending on the question, the instance data, and the solver. Also because of the semantic gap between operations research analysis models such as job-shop scheduling algorithms and IT-based models such as manufacturing execution systems (MES), there is little integration between control analysis methods and control execution tools. Automated and cost-effective access to multiple analyses from a single conceptual model of the target system would broaden the usage and implementation of analysis-based decision support and system optimization.

The fundamental contribution of this dissertation is concerned with interoperability and bridging the gap between operations research analysis models and practical applications of the results. This dissertation closes this gap by constructing a standard domain-specific

language, standard problem definitions, and a standard analysis methodology to answer the control questions and execute the prescribed control actions.

The domain specific language meets a broader requirement for facilitating interoperability for DELS, including system integration, plug-and-play analysis methods and tools, and system design methodologies. The domain-specific language formalizes a recurring product, process, resource, and facility description of the DELS domain. It provides a common language to discuss our systems, including the questions that we want to ask about our systems, the problems that we need to solve in order to answer those questions, and the mechanisms to deploy the solution.

A canonical set of control questions defines the comprehensive functional specification of all the decision-making mechanisms that a controller needs to be able to provide; i.e. a model of analysis models or a metamodel of operational control. These questions refine the interoperability mechanism between system and analysis models by mapping classes of control analysis models to implementation and execution mechanisms in the system model.

A standard representation of each class of control problems is only a partial solution to fully addressing operational control. The final contribution of this dissertation constructs a round-trip analysis methodology that completes the bridge between operations research analysis models and deployable control mechanisms. This contribution formalizes an analysis pathway, from formulating an analysis model to executing a control action, that is grounded in a more fundamental insight into how analysis methods are executed to support operational control decision-making.

CHAPTER I

INTRODUCTION TO OPERATIONAL CONTROL FOR DISCRETE EVENT LOGISTICS SYSTEMS

1.1 What Are Discrete Event Logistics Systems?

Discrete Event Logistics Systems (DELS) are a class of dynamic systems that are defined by the transformation of discrete flows through a network of interconnected subsystems [190]. As a discrete event system [220], DELS evolve in response to random, and often unpredictable, discrete events such as arrival of a customer, completion of a processing task, or pre-emptive equipment failures. DELS are inherently complex systems due to the large scale of the networks, the dynamic nature of interactions between actors, and the randomness of both external and internal environments.

The DELS domain encompasses a diverse collection of systems that are generally regarded as very different from one another and thus require dedicated research tracks. Each member of the DELS domain shares fundamental characteristics, but adds its own flavor in terms of domain specific semantics, additional constraints, capability requirements, etc. Each also lives in its own unique ecosystem that generates unique events that the system must respond to and conditions that affect the evolution of the system.

The traditional subclasses of DELS are supply chain systems, manufacturing systems, transportation systems, and material handling systems [267]. The following are a non-comprehensive list of other members of the DELS domain and what makes them unique:

1. Healthcare logistics (derived from manufacturing) have unpredictable and unique demand, which generates unique process plans on a case-by-case basis, e.g. treatment of cancer versus trauma; and the product mix of elective vs emergency can be unpredictable [219].

2. Humanitarian Logistics operates in a complex ecosystem that requires preparing for

and responding to sudden, diverse, and geographically dispersed events with challenging last-mile distribution conditions [11, 159].

3. Semiconductor manufacturing has short product life-cycle and high product mix and process planning deals with re-entrant flows and tool dedication among many other processing restrictions [283, 189].

4. Reverse Logistics [72] extends the standard supply chain by incorporating additional functions to retrieve, return, and retire goods at the end of their lifecycle.

5. Remanufacturing [118] incorporates the uncertainty of timing, quantity, and quality (yield of material recovery) of returns. The processing component faces stochastic process plans (routing) and high uncertain processing times.

Fundamentally, these systems exhibit many common structural and behavioral characteristics. One such characteristic, humans, also happens to be one of the most challenging aspects that differentiates these systems from other engineered dynamic systems. Each of these systems has humans as part of the system, which makes the system more complex than can be modeled with equations of motion.

1.2 Why Control?

The design and analysis literature for the DELS domain focuses on the types of operations research analysis models that can be solved to produce a system design or control specification [27, 187]. Rouse [228] provides a context for this body of literature by defining several tiers of analysis:

1. *Describe* past observations

2. *Classify* past observations

3. *Predict* future observations

4. *Control* future observations

5. *Design* future observations

While there has been substantial research in the field of "big data" on developing real-time diagnostic and predictive analytics [145], the natural evolution of leveraging information is toward prescriptive analytics, that is how to make decisions regarding the "best" way to operate a system and control its performance [44]. Apte [12] subscribes to the same natural evolution of leveraging information and elaborates the definition of prescriptive analytics:

> "Prescriptive analytics allows a user to obtain an actionable solution, getting an answer to the question, 'what is the set of required actions' to take to achieve a business objective, under a given set of predictions and business constraints" [12].

The prevailing paradigm in the related operations research literature neglects to conceptually or operationally separate the model of the plant from the model of the control of that plant. The result of this lack of separation is that the research, design, and verification of the system components is done in isolation; e.g. optimal control models use implicit system models with simplified or ideal behaviors, or resource investment models assume a static control policy. Ultimately, this isolated development is both the result of and leads to limited interoperability and integration between system models and analysis models, where currently the translation of academic research into deployable solutions is done by hand. A clear definition of and language for operational control would enable analysis interoperability and libraries of plug-and-play analysis tools. For example, separating the control mechanism and encapsulating the control policy makes the policies themselves interchangeable (strategy pattern, [102]); therefore the policy can be modified in order to dynamically change the strategy for managing the system, without changing the underlying implementation of the policy [257].

1.3 What is Control of DELS?

A large portion of the research that is conducted in the realm of DELS is related to the the control of these systems, but there are many different ways to define the many different control activities that occur in a DELS. Traditionally, the planning and control aspects

of operations research are divided into three groups reflecting their respective planning horizons: strategic, tactical, and operational [266]. SCOR's processes reflect a similar organization: plan, enable, execute/do [244]; which as a process model neglects the systems and resources executing the activities. While Mönch et al. [190] extends the enterprise model by adding a base subsystem consisting of the resource and working objects, details are sparse on how control is implemented at each level.

Extending our system view beyond the domain of operations research, ISA-95 is an international standard for the integration of enterprise and control systems. ISA-95 consists of models and terminology that can be used to determine which information has to be exchanged between systems for sales, finance and logistics and systems for production, maintenance and quality. ISA-95 separates the manufacturing domain into four levels: 4) Business Planning & Logistics (ERP systems), 3) Manufacturing Operations Management (MES, WMS) 2) Manufacturing Control Systems (PLC, DCS, SCADA), and 1) Intelligent devices.

Manufacturing Control Systems: Execution-Level Control - ISA-95 Levels 1 and 2 The research on implementation-level details and development of distributed industrial control systems standards, such as IEC-61131, have focused on enabling real-time monitoring and execution-level control of large-scale industrial processes (ISA-95 Levels 1 and 2). This led to a significant amount of effort being put into designing and managing these complex systems, including research in topics such as: how is the control network organized [80], how control networks should be implemented [SCADA, [99]], how to generate legal sequences of controller actions [MPSG, [260]], generating PLC code [IEC standards, [295]], and use of automata and formal language theory to derive the existence and structure of controllers, and define a controllable language for discrete event systems [220].

Manufacturing Operations Management - ISA-95 Level 3 The field of operations research is primarily focused on prescriptive analytics, where the notion of control has a different flavor than that implied by the automated manufacturing systems literature. Research at this level is the intersection of domain problems and solution methods and does

not focus on implementation details, but is diverse nonetheless; e.g., routing and scheduling of vehicles and crews [218], machine scheduling [110], warehouse storage allocation [128]. Whereas ISA-95 Level 2 standards provide the capability to turn a server on and off, optimal control models try to find the optimal operating policy, that is, rules for turning the server on and off that result in the lowest low-run cost [270]. This distinction separates the control responsibilities between ISA-95 Levels 2 and 3. Common tools for executing operational control at this level are manufacturing execution systems (MES), transportation management systems (TMS), and warehouse management and control systems (WMS/WCS).

Business Planning & Logistics - ISA-95 Level 4 ISA-95 Level 4 defines the business-related activities needed to manage a manufacturing organization. These activities do necessarily correlate to long-term strategic planning activities, but rather focus on establishing the plant production scheduling, from scheduling material consumption through product delivery and shipping. In concert with the production scheduling, this level is responsible for establishing resource levels to support production activities, include setting inventory and staffing levels. Research at this level includes problems such as inventory investment and allocation [56], newsvendor problems [152], collaborative planning, forecasting, and replenishment (CPFR) [96], and tools such as enterprise resource planning (ERP) systems. It does not appear that this level covers strategic planning activities such as facility location problems [104].

1.3.1 Modeling Control

An important aspect of designing the control of a dynamic system is deciding what information from the system is used to make control decisions. In an open-loop controller, adjustments are made without ever receiving feedback from the system. This requires the controller to have a perfect model of the device and environment, as adjustments to compensate for inaccuracies are not available. Closed-loop policies can incorporate the additional information generated in each time period into the decision making process. There are two types of closed-loop controllers: feedback and feed-forward. In a feedback controller, the controlled variable is fed back into the controller. The controller relies on measuring the

variable and making adjustments. In a feed-forward controller, disturbances are measured and accounted for before they impact the system.

In DELS, control strategies are often designed to use a combination of all three. Open-loop control is expressed in non-adaptive control policies such as *a priori* sequences of routing or dispatch that do not incorporate feedback into their control decisions. Material requirements planning (MRP) sets a fixed production schedule for a specific horizon and does not respond to the resolution of stochastic events, such as production delays, until a pre-defined re-planning event (an example of open-loop controller embedded in a larger closed-loop controller). Closed-loop feedback control is used in examples such as turning on an additional machine in response to long cycle times or using inventory levels to initiate replenishment orders (inventory policy). Finally, the forecasting mechanism embedded within a replenishment policy is a feed-forward control loop [71].

The basic problem in supervisory control is to modify the behavior of a given discrete event system so that it lies within some prescribed range [220]. This desirable range may be specified by actually giving the desired closed loop behavior, by giving a behavior within which the closed loop behavior must be contained, or by specifying such sets indirectly through other qualitative performance objectives. In the logic controller approach [310], the controller must translate commands from an external agent into a sequence of operations to be performed by the plant. In controlled Markov chains [44], action is taken to control the probabilities that affect the evolution of the chain. Finally, controlled petri nets are a class of petri nets with external enabling conditions called control places which allow an external controller to influence the progression of tokens in the net [137]. Then in general, control is a sequence of actions taken to affect the evolution of the state of the system in a manner that is optimal with respect to a set of objectives.

However for controlled petri nets, finite state machines, and abstract state machines, each formalism requires some external function to be applied to the system to help it evolve. Controlled petri nets allow an external controller to influence the progression of tokens in the net [137]. The switch place of an Extended Petri Nets uses information, or "control logic", external to the petri net model to resolve conflicts that arise [at the switch place].

Abstract State Machines can implement a collection of 'next-state functions' that would define a set of control functions for the domain and express how the transition system evolves [223]. There is not a definitive way to construct these functions, and they "follow no general rules, but depend on the logic of the specific problem at hand and the desired system behavior" [284].

While implementing supervisory control requires some external decision process to be applied to the system to guide its evolution, there is no formal, canonical specification or design methodology for this external process. However, due to the dynamic nature of DELS, the design of optimal control models shares many characteristics with engineering other dynamical systems. This has led to the incorporation of theories and mathematical tools from control theory to support the specification and optimization of dynamic aspects of DELS behavior [237].

In control theory, the controller uses the reference point and measured output from the system to determine a control decision or input to the system (Figure 1). In classical control theory, the controller is implemented using transfer functions based in the frequency domain. However as the systems evolved, it became difficult to design and deploy more complicated controller architectures such as multiple input, multiple output (MIMO) control methods. In modern control engineering, the physical system is modeled as a set of input, output, and state variables related by first-order differential equations, a form known as the state-space representation. The set of state variables is defined as the minimal subset of system variables that can represent the entire state of the system at any given time [195].

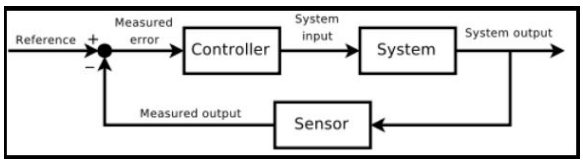


Figure 1: Classic Control Theory Block Diagram

Whereas in classic control theory the controller's design is static, adaptive control is a technique that allows the parameters of the controller to be adjusted using feedback obtained during the operation of the plant [239]. Sastry and Bodson [239] informally define

adaptive control as "a direct aggregation of a (non-adaptive) control methodology with some form of recursive system identification", which allows the controller to progressively improve its understanding of the system. In model reference adaptive control schemes (Figure 2), the desired performance of the closed loop system is specified through a reference model [299, 167]. An additional feedback loop, the *outer loop*, allows the adjustment mechanism to adjust the parameters of the controller to match the plant's output to the reference model.

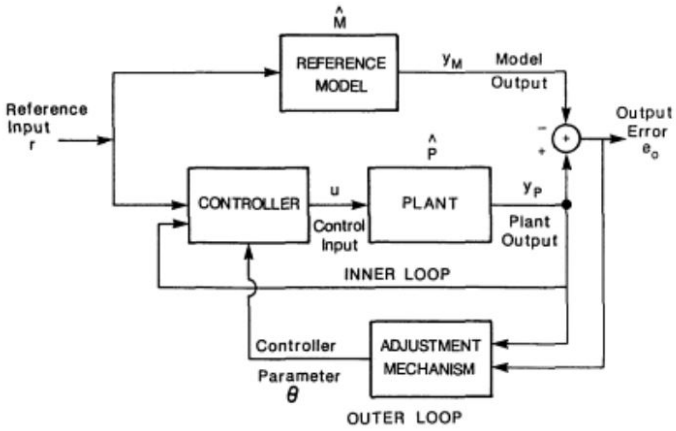


Figure 2: Model Reference Adaptive Control Block Diagram [239]

Simon [255] was the first to apply control engineering theories to production and inventory control. In this model (Figure 3), the optimum inventory level θ_I is taken as the reference point, the actual inventory of finished goods, θ_O, is the system output, and ϵ measures the inventory deficiency (both positive and negative). This deficiency is input into the decision rule K_2 that determines the production level in the period, $\mu(t) = K_2[\epsilon(t)]$. The customers' orders θ_L are then subtracted from the period's production, $\theta_O = K_1[\mu(t) - \theta_L(t)]$. The function K_2 is a decision policy that specifies a rate of production as a function of the excess and shortage of inventory. However it follows from classical control theory that both K_1 and K_2 are assumed to be linear operators.

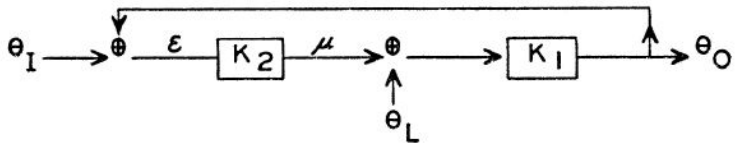


Figure 3: Control Theory for Production and Inventory Control [255]

Over the next half century, this control-theoretic approach has been elaborated to support more complex system dynamics, see [81] for an overview of the research. In Figure 4, Disney and Towill [81] extend the basic model from Simon to support more elaborate models that incorporate multiple effects, including disruptions and forecasting, to optimize production levels. Despite the additional complexity, there remains a fundamental controller structure that uses inputs to the system (demand pattern) and a decision rule to set the output (replenishment or production orders) from the system. Each replenishment rule is modeled by a corresponding transfer function that completely represents the dynamics of the rule. However, transfer functions limit the types of decision rules that can be implemented, e.g. inventory and production levels rather than detailed scheduling. For another line of research in this area, see [243].

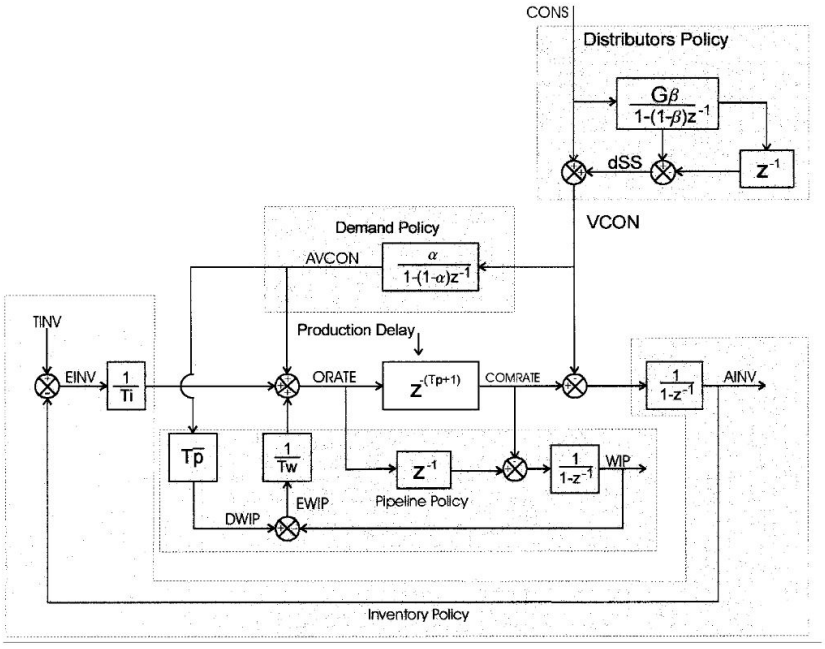


Figure 4: Control Theory for Production and Inventory Control [81]

However, Bona et al. [32] made a simple, yet important extension to this paradigm by implementing the control theoretic framework without transfer functions, but instead using scheduling algorithms in their place. Their model exhibits many characteristics of the control-theoretic approach including an explicit system model, a reference point (the proposed schedule), and a decision rule or more complex rule-based control. Also, the system model incorporates complexity and uncertainty that requires the system to modify and adapt the proposed schedule. In this model, the control is a function that takes the state of the shop floor (the actual production as a proxy) and maps it to a set of control actions that modify the schedule, i.e. action = *f(*state).

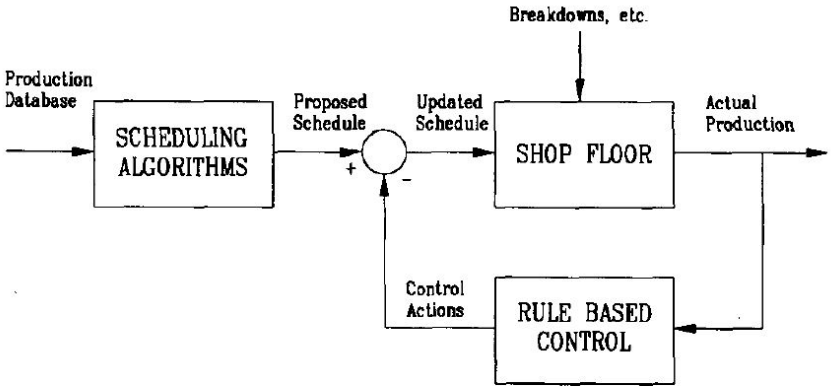


Figure 5: Control Theory for Production and Inventory Control [32]

These models provide a clear evolution of formal models of control from linear transfer models to more general forms that support rule-based and more advanced control mechanisms. There remains a need to generalize the model further to extend control to support the entire range of control decisions that must be made at the operational level of control in DELS. In the remaining discussion in this introductory section, a mental model for this generalized control model is constructed as a direct descendant of the work presented above.

This mental model (figure 6) explicitly recognizes a need for the controller to be specified separate from the model of the plant **(0)**. Ignoring pure open-loop control cases, **(1)** the controller must interface with the plant or base system to receive or extract system feedback. However, while technical and syntactical interoperability may be sufficient, ideally there needs to be a consistent representation of the state of the plant **(2)**, which will be the

target of developing a DELS domain-specific language. To guide the trajectory of the system, the controller needs a planned trajectory in the form of requirements, goals, utility, etc. **(4)**. The controller must internally compare and evaluate the current trajectory versus the expected trajectory and select an action to execute **(3)**. This action may change the future trajectory of the base system to conform to the planned one, or change/request to change the planned trajectory. The evaluation function in **(3)** may incorporate additional information aside from the current state of the system into the decision making process, including past/observed system information, projected system states, and the expected trajectory of the system.

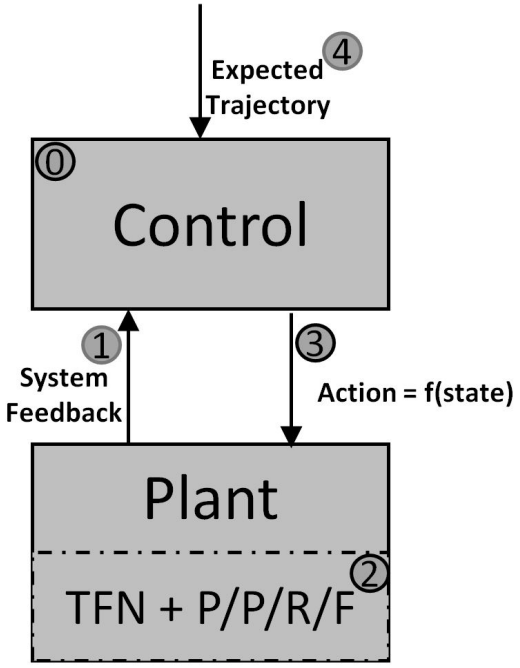


Figure 6: A Mental Model of the Separation of the Controller from the Plant

1.4 What do we need?

There is a wide variety of definitions of the control activities in DELS, but the research is overwhelmingly focused on implementation-level control in the manufacturing systems domain. Moreover, computer integrated and automated manufacturing technologies are more mature than automation in other domains, such as material handling or logistics. Integration of these systems will become a more important issue as autonomous devices

become more common-place in the broader DELS ecosystem. In order to be able to design the control mechanisms of these systems, we need to be able to describe the control activities that are being executed at the operational management level.

With integration and interoperability in mind from a design perspective, there is a need for a uniform representation of the plant and controller structure for each DELS. A uniform controller architecture enables both the integration of diverse DELS components, but also enables a diverse spectrum of control architectures to be instantiated from a uniform language and reference architecture. From this perspective, each DELS controller needs the functional capability to make every control decision, which requires a comprehensive set of control analysis models and execution mechanisms.

Then one key aspect to evolving the state of the art of DELS controllers is automated and cost-effective access to a wide variety of analysis methods to support the control questions that need to be answered to make decisions. The field of operations research supports control decision-making (prescriptively) at the operational management level by specifying how control decisions should to be made. However due to a semantic gap between operations research models and IT-driven models [187] such as MES, WMS, and TMS; there is little integration between these analysis methods and execution tools.

Therefore there remains a need for a more comprehensive model that incorporates the control activities at the operational level of control, bridges between analysis models and mechanisms to implement control decisions, and finally, extends the scope to support the broader class of DELS.

1.5 Contributions

To support the development a controller architecture that generalizes the ISA-95/L3 for all DELS and meets the requirements stated above, this dissertation constructs a more general model, a metamodel of operational control for DELS, that implies the ISA-95/L3 controller architecture as a special case, or usage of the metamodel. This metamodel requires three components: a language to describe the operation control problems themselves and the DELS system model they are operating on; a specification of the operational control

problems; and a complete, round-trip analysis methodology for solving these problems and executing the prescribed control action.

The first contribution of this dissertation is a domain-specific language (DSL) for describing the behavior of DELS. The DSL formalizes a product, process, resource, and facility (PPRF) pattern that underlies every system description in the DELS domain. This language extends the token flow network (TFN) description for specifying the structure of DELS [273], and establishes a common language to specify and discuss the systems of interest, specifically the questions we want to to ask about our systems, the problems that we need to solve to answer those questions, and the mechanisms to deploy the solutions. This domain-specific language meets a broader requirement for accomplishing interoperability for the DELS domain, including system integration, interoperable or plug-and-play analysis models and tools, and DELS design methodologies.

While the DSL fulfills a more general interoperability objective, control questions refine and extend the interoperability mechanism between system and analysis models by formulating a canonical functional description of operational control. This fundamental set of control questions is formalized by a standard abstraction formulated using the abstract PPRF semantics. The questions themselves form a comprehensive functional specification of all the decision-making mechanisms that a controller needs to be able to provide; i.e. a model of analysis models or a metamodel of operational control.

The third contribution of this dissertation addresses the need for round-trip analysis capabilities for operational control that completes the bridge between operations research analysis models and deployable control mechanisms; that is, not only the ability to formulate and solve operational control problems, but also to translate the results into policies, plans, and executable actions. This round-trip analysis methodology is grounded in a more fundamental insight into how analysis is executed for operational control decision-making and prescribes a functional architecture for a DELS controllers.

These contributions are the necessary components to construct uniform controller architectures for operational control of DELS, which allow for both the integration of adjacent DELS domains but also the development of a flexible and extensible controller architecture

that can extend from centralized, hierarchical control to distributed, agent-based control.

CHAPTER II

A DOMAIN SPECIFIC LANGUAGE FOR DELS

Product definition languages for manufacturing systems have enabled partial interoperability to be achieved between the various engineering tools, such as computer aided design (CAD), product data management (PDM), and product lifecycle management (PLM), that are used to design products. Process definitions have become more important as manufacturability has become an integral aspect of product design. However, these standards only apply or have been applied to manufacturing and are not intended, nor are semantically complete enough, to specify system models and construct corresponding analysis tools for the broader domain of discrete event logistics systems.

Formal domain modeling approaches have been applied extensively in manufacturing systems for more than thirty years, see e.g. [304, 263, 285, 260, 264, 184, 172]. Furthermore, Grubic and Fan [114] provide an overview of supply chain ontologies, including IDEON [177], TOVE [97], and the manufacturing system engineering (MSE) ontology [173]. Lin et al. [173] also review some early manufacturing system information models.

Each of these formal system models exhibit a recurring pattern: a product, process, resource, and facility (PPRF) description of the system of interest. This PPRF pattern is not limited to manufacturing systems and is common throughout the DELS literature. For example, the design of the overall structure of the warehouse facility includes decisions such as department layout, sizing, and dimensioning; as well as "how many storage departments, employing what technologies [resources], and how [what process] orders [products] will be assembled" [116]. Across a broad spectrum of operational scenarios, health care systems seek to provide a proper course of treatment (process) to each patient (product) using a diverse array of resources, such as nurses, doctors, operating rooms, and ambulances; that "matches the medical requirements, capacity requirements and restrictions, and the facility's layout" [208, 219, 142]. Humanitarian and disaster relief logistics systems distribute products,

and sometimes resources such as vehicles, using a limited set of transportation resources [95, 200, 294]. Commonly these problems formulate distribution plans (process sequences) which sequence the pick-ups from supply depots (facility) and drop-offs of products to areas of need.

To bridge the gap between system descriptions and mathematical programming analysis formulations, Powell et al. [210] argue the similarity of broad classes of operations research problems, termed dynamic resource transformation problems, and seek to provide an abstract language or mathematical formulation of these problems.

> "Our ability to solve these problems is limited by the languages that we use to express them. ... clearly, this tendency [to view problems as vastly different] reduces our ability to learn from similar problems in different industries and excessively fragments the field. ... in this paper, we do not focus on an algorithm for solving a problem but rather on simply representing the problem in a general way that captures the much richer set of modeling issues that arise in a dynamic setting." [210]

However, mathematical abstractions of domain-specific problems have a limited ability to bridge the gap between analysis models and deployable solutions. To the best of our knowledge no one has attempted to provide an object-oriented formal language that is broadly applicable to the DELS domain, and there remains a requirement for a standard description of the DELS behavior.

This dissertation will formalize the product, process, resource, facility (PPRF) pattern that is a recurring description of the behavior of these systems to create a object-oriented language for creating valid DELS class definitions and conforming system models. This language is constructed as an extension of the structural semantics provided by the token flow network (TFN) [273] While an axiomatic DELS definition is beyond the scope of this dissertation, a semantically precise language for DELS enables progress towards interoperability between system models and analysis models and integration of systems models from adjacent domains, such as transportation logistics and warehouse material handling

systems. A common modeling language introduces a natural mapping between the abstract mathematical description of solution algorithms and the system models that provide the necessary context and data, and is critical to bridging the gap between these analysis models and practical implementations of their corresponding tools. While there may exist other frameworks capable of specifying DELS models and achieving the stated interoperability goals, what we seek to demonstrate is that a DELS language that formalizes the PPRF pattern provides the minimal, complete set of domain-specific semantics required to describe DELS models.

2.1 Product

In the manufacturing environment, a product is defined by a bill of material (BOM) and a process plan, i.e., assembling this list of materials per this process plan will result in the desired product. In the warehousing environment, the product can be defined similarly as a pick list and the process plan specifies a route to all the required storage locations. However in transportation logistics, the product is the same as the input except its geographic location has been transformed. Similarly by storing an object, its age has been transformed. The common idea across all of these system descriptions is that the product is what is flowing through and being transformed by the system.

Formally, the *Product* is represented by a token that is output from top-level *Process*(es) executed by the DELS (figure 7): "an enterprise's products are a subset of all the outputs from the enterprises processes. A product can be a physical product [*isPhysical*] that can be sold to the customers, a document, a service to the customers, an executable process (or skeletal plan), or a new information system" [177] (figure 8). The Process that creates the Product specifies the set of inputs —resources, raw materials, and information —required to create the Product, and the bill of material (BOM) can be specified using the *requiredInputResources* attribute.

Product can also be designated as *isAggregate*, which implies that the product is the aggregation of other products included in its *aggregatedContents*. These aggregatedContents are not necessarily constrained to be a homogeneous set (*hasHomogeneousContents*). For

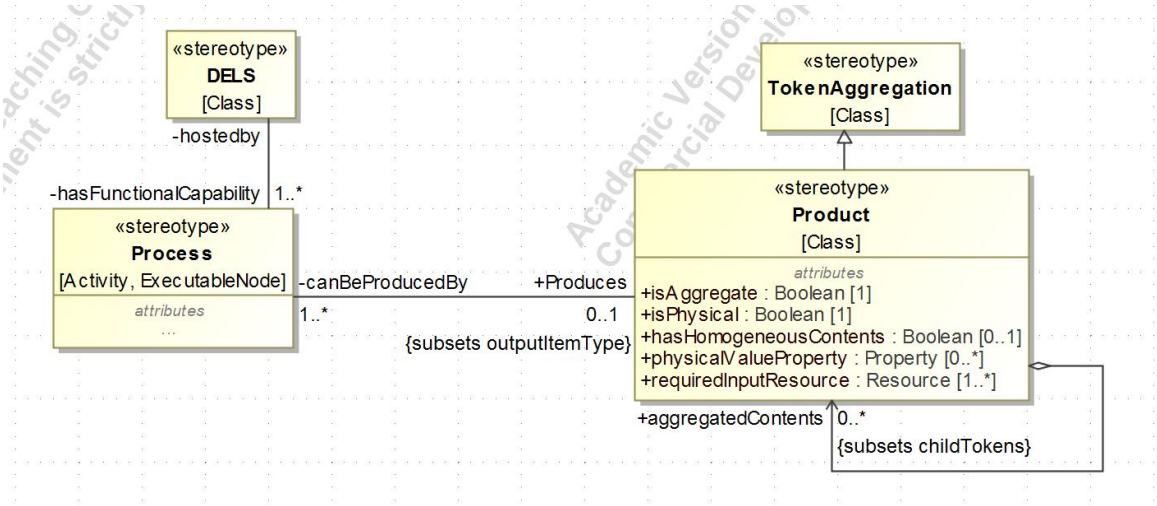


Figure 7: The Product is the output of the top-level Process(es) executed by the DELS.

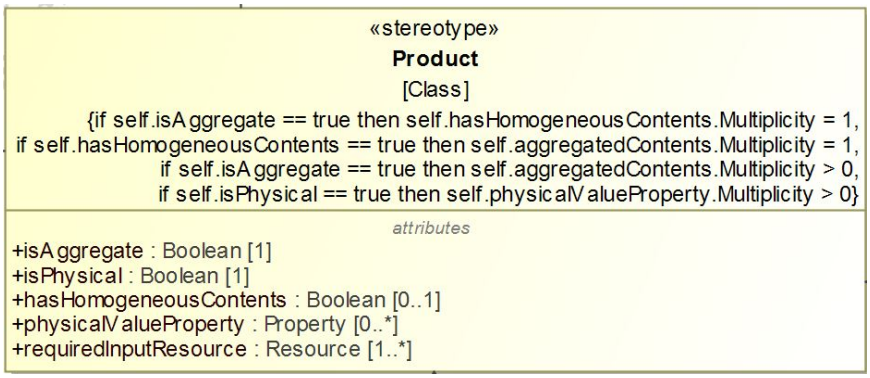


Figure 8: The contraints defined on the Product stereotype define well-formed rules for constructing valid Product classes.

example in figure 9, a shipment from a warehouse is the output from the *SHIP* process and is the aggregation of a (not necessarily homogeneous) set of stock keeping units (SKUs). In this role, the SKUs are viewed as the inputResourceType that goes into the SHIP process that produces the shipment (figure 10). However, aggregation conveys an important attribute of the shipment because this shipment can be disassembled in the future and the SKUs would retain their Product identity.

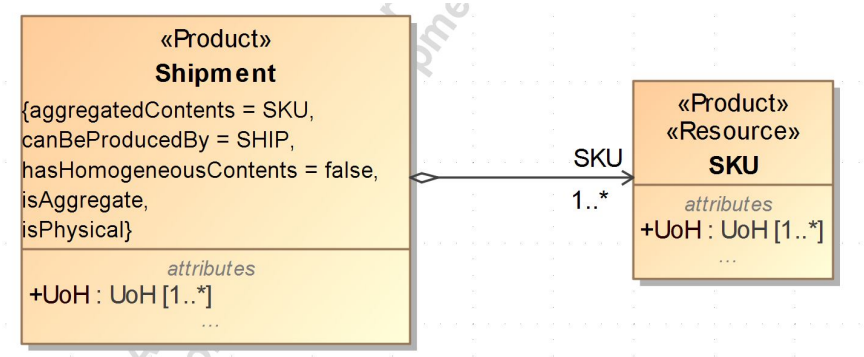


Figure 9: A warehouse shipment is an aggregate product that contains SKUs.

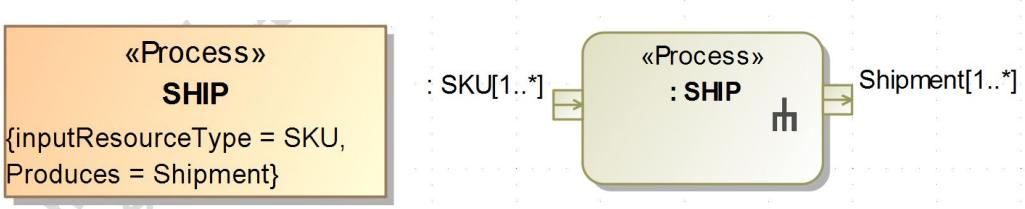


Figure 10: The Shipment is the output of the SHIP process and the SKUs are both the inputs to the SHIP process and the Aggregated Contents of the Shipment.

The *isAggregate* and *hasHomogeneousContents* attributes of the Product stereotype will also be necessary when Resources are discussed in section 2.3 to connect the output of one process to the required inputs of a subsequent process. Finally, if the Product *isPhysical* then a set of *physicalValueProperties*, such as weight or volume, *must* be specified.

2.1.1 Representation Methods for DELS' Products

In manufacturing, the bill of material (BOM) captures all of the raw materials and sub-components, i.e., products of other procurement or manufacturing processes, that are required to produce the product. Hu et al. [140] provide an overview of assembly representation methods and discuss the limitations of current representations to represent complex

physical assembly processes. The product architecture in [281] defines the schema to associate physical components to functional elements to form different products. Hegge and Wortmann [130] describe a generic BOM model that uses object-oriented modeling methods to capture functional and structural relationships among product sub-components. The Core Product Model extends this functional view of the product to include its function, its form, and its behavior to capture information relevant to PLM process [94].

Zha et al. [308] conducted a comprehensive review on graph-based assembly representations, which represent assembly entities as vertices on a graph. Jiao et al. [147] develop a generic Bill-of-Materials-and-Operations, a data structure that integrates the bill of material and the bill of process as a bipartite tree/graph. The authors also incorporate variant management into their data structure through AND/OR digraph semantics in the BOM. However, Tursi et al. [280] make the case that the BOM isn't the only piece of information that is required for a complete description of a product.

With the maturity of computer aid engineering methods, technologies for capturing the product specification such as PDM and PLM are more mature and integrated into the manufacturing engineering methods than in other fields. Building upon the ISO 10303 and IEC 62264 standards, Tursi et al. [280] and Panetto et al. [202] define a Product Ontology to formalize the technical data and concepts associated with a Product. While not as mature and well-studied as the manufacturing use case, health care ontologies are being applied to personalize the medical and social knowledge available for a patient with a particular condition [224].

Knowledge-based product models that incorporate object-oriented programming (OOP) methods and artificial intelligence (AI) techniques offer the potential to formally capture tacit human expertise [308]. In holonic and agent-based manufacturing architectures, such as PROSA [285], this knowledge is deployed by the product holon (or agent) to negotiate and coordinate (routing) its path through the system.

Bill of material (BOM) matrices are the most common mathematical representation for

capturing product knowledge in math programming formulations involving product assembly problems:

$$A = \{a_{ij}\},\ \forall i \in \{1, 2, \ldots, N\}$$

where a_{ij} is the quantity of item i required to produce one unit of item j. If E is the set of 'end products', I is the set of 'intermediate products', v_i is the set of 'successors to i', and w_i is the set of 'predecessors of i', then:

$$E = \{i \mid a_{ij} = 0,\ i = 1, 2, \ldots, N, j = 1, 2, \ldots, N\}$$

$$I = \{i \mid \exists\ 1 \leq j \leq N \text{ s.t. } a_{ij} > 0,\ i = 1, 2, \ldots, N\}$$

$$v_i = \{j \mid a_{ij} > 0,\ j = 1, 2, \ldots, N\}$$

$$w_i = \{j \mid a_{ji} > 0,\ j = 1, 2, \ldots, N\}$$

2.1.2 Applications

In many DELS descriptions in the literature, the product definition is usually implicit. In warehouses and other storage systems, the primary function is to store their product for a period of time [229], but they also may have auxiliary services such as kitting and constructing mixed-pallets for shipment which add value to the resulting product. In health care systems, often the process plan must be customized to the unique requirements of the patient [142]; this is very similar to the planning of engineer-to-order and one-off manufacturing products. In humanitarian and disaster relief logistics, vehicles can be treated as commodities that accompany the actual commodities, such as "medical materials and personnel, specialized rescue equipment, and rescue teams, food, etc." [200]. In all of these applications where a multi-commodity flow network model is formulated, an intermediate abstraction step is required to map the content of the product to an abstract commodity.

In [165], the system constructs a set inventory packages, which are collections of inventory types, and then associates each inventory package to a price(s) to form a fare product. This inventory package-fare product combination is referred to as the product. While this model is constructed for the airline, vacation package, etc. domain, it may also applied to

kitting services and assembling mixed pallets of inventory goods in a warehouse for grocery stores.

Aside from their manufacturing origins, BOM and other product family representation methods are a useful way to describe the product in adjacent DELS domains. In warehousing, shipments may contain a bill of lading that details each of the products contained in the shipment. Similarly, disaster relief systems may maintain a standard product family(ies) by "keeping standard aid kits in regional warehouses and distributing them to local hubs at the onset of a disaster with minor adjustments depending on the local needs and specific demands of the disaster" [289].

While several standards exist for describing manufactured products, there is a gap between the specification of the product and the specification of the production of the product. The Product class in the DELS DSL provides a specification of the product that integrates with the production system models and analysis models. Furthermore, this description extends to support the specification of products in domains that do not explicitly manufacture the product, and integrates that specification with DELS analysis models and tools.

2.2 Process

The set of *Processes* that a DELS hosts defines the DELS's *functionalCapabilities*. Process in the DELS language extends and refines the token flow network (TFN) definition (figure 11). By extending the element rather than simply reusing it and adding DELS elements as necessary, this approach keeps the TFN self-contained and abstractly focused on token flows and does not clutter the language with DELS semantics, such as product and resource flows. In this section, the Process stereotype in the DELS language will be extended to include relationships to products and resources.

In the TFN definition, the Process defines *inputItemType*s and *outputItemType*s, which are types of tokens that can flow in and out of the process node, respectively. In the DELS definition (figure 12), a subset of the inputItemTypes are *inputResourceTypes* which are the Resources required to execute the Process, and a subset of the outputItemTypes is the *Product* that the Process produces. The Product and Resource will eventually be used to

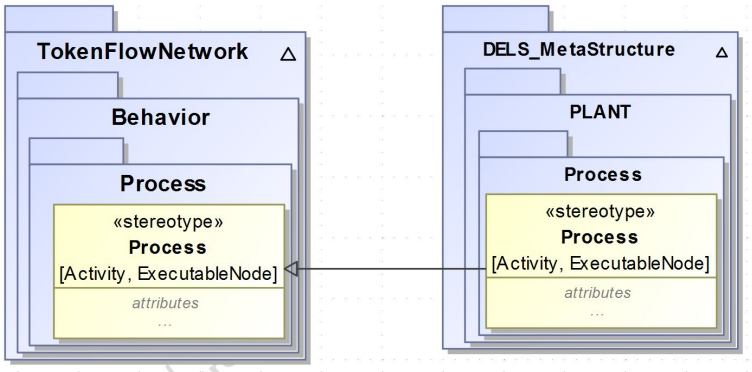


Figure 11: The Process stereotype in DELS language refines and extends the Process stereotype from the TFN.

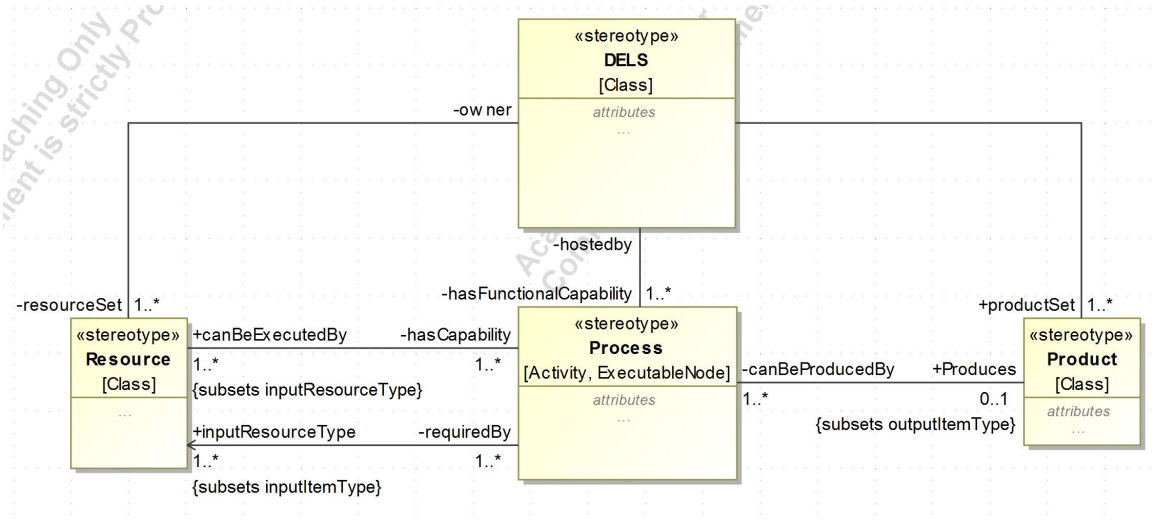


Figure 12: A DELS' functional capability is defined by the Processes that its ResourceSet can perform.

type the Input and Output pins (or interface Flow Nodes) of the Process activity.

Many system descriptions in the literature delineate between executing resources, e.g. machine and operator, and more general or basic input resources, e.g. raw materials or subcomponents. Therefore, contained within the set of input Resources is a set of Resources that will be defined as the executing resources (*canBeExecutedBy*), which have their capability set defined by the Processes they can execute, cf. section 2.3. The execution semantics for process networks are similar to the execution of petri nets, where a process cannot execute until all required tokens, resources or otherwise, are present.

UML activities are particularly limited in their expressiveness of process plans. The

biggest reason is that there is one less layer of instantiation or usage; that is, the M0 level represents the run-time execution of a particular activity diagram and any particular pathway a token may take. The Process Specification Language [242] separates the definitions of activity and activity occurrence, which alleviates the semantic issues related to the activity/action/execution semantics in UML. Therefore, it should be a long-term goal to extend the UML activity definition language to support the complexity of process plan specification included in PSL. For the purposes of this dissertation, Process will be defined formally using the process node extended from the TFN definition and informally merged with PSL semantics for discussion purposes.

2.2.1 Process Plan Formalisms

Process plans are a standard way of organizing the execution of processes in DELS. Much of the work on defining planning and scheduling formalisms comes from the manufacturing literature, where the "process plan representations must exhibit the following capabilities: explicit parallel and alternate sequences, multi-job synchronization, hierarchical task decomposition, resource management primitives, and user extensibility" [246].

In project scheduling, the structure of the project often is depicted by an activity-on-node network, where the nodes and the arcs represent the activities and the precedence relations, respectively [38, 216]. Depending on the application, activity-on-arc models can be employed as well [141]. Disjunctive graphs have been used in job-shop scheduling problems because of their ability to capture processing alternatives in multi-processor environments [19, 63, 155, 39]. Planning and scheduling models based on the disjunctive graph formulation are generally attributed to [230]. AND/OR digraphs extend the disjunctive graph semantics by defining alternative task and sequence requirements using OR junctions to represent alternative paths and AND junctions for parallel paths without specifying a particular execution sequence [138, 45, 304]. Homem de Mello and Sanderson [138] provide a method to generate and evaluate possible assembly sequences as an AND/OR digraphs from a model of the Product.

Wysk and Smith [304] demonstrate several important advantages of representing process

plans as digraphs. First, each node nests its own digraph representing the decomposition of the process into smaller, and eventually atomic, processes. Second, they present a process to produce a serialized process list from the digraph, which is their definition of the planning and scheduling problem. Third, they capture the duality of a Product traversing its process plan as a control graph that formalizes the processing requirements of all the tasks to be processed by a controller.

There are at least two formal language definitions for process specification, A Language for Process Specification (ALPS) [45] and the Process Specification Language (PSL) [242]. ALPS refines the AND/OR nodes by adding predicate functions or paramaterization to indicate eligibility for specific paths, both of which are example of guards on branching behavior. The PSL is of particular interest, not only because of its popularity but also because of the comprehensive nature of the specification. There are nine extensions to the PSL Core language that are relevant to our modeling efforts, four dealing with generic process modeling and five that deal with schedules (figure 13). Cheng et al. [52] apply the PSL to [construction] project scheduling, because "generally speaking, PSL has more expressive power than many project management tools".

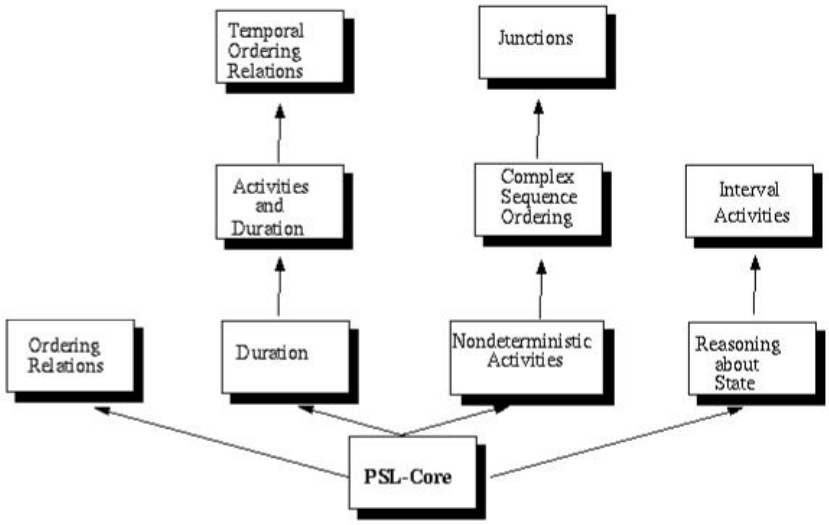


Figure 13: PSL Extensions for Generic Activities, Ordering Relations, and Schedules [242]

The activity-on-node network representation is the most common general, mathematical representation of a process plan. In this representation, the nodes and arcs represent the

process, or activities, and the precedence relations, respectively. The set of nodes and arcs are specified by the graph structure $\mathcal{G} = (V, E)$. The precedence constraints are denoted by $i \rightarrow j$ or (i, j) and $Pred(j) = \{i \mid \exists (i, j) \in E\}$ defines the set of direct predecessors of the process j, and likewise, $Succ(j) = \{i \mid \exists (j, i) \in E\}$ defines the set of direct successors of the process j. We may also consider a nested process plan, $\mathcal{G}_j = (V(j), E(j))$ as the set of nodes and edges nested within the node j; however, this representation can be flattened and specified as a one level process plan. We refer the reader to [201] for a formulation of the flexible job shop scheduling problem with process plan flexibility, where each job has multiple process plans to choose from and multiple machines capable of processing each operation.

2.2.2 Applications

Formal process specification languages can used in a diverse array of applications in the DELS domain where process plans are constructed, either explicitly or implicitly. These specification languages are especially useful in capturing complex process plans that incorporate process and resource flexibility. For example while the job-shop scheduling problem [110] typically assumes only one available machine for each operation and one feasible process plan (operation sequence) for each job, Özgüven et al. [201] relax these two assumptions by considering routing flexibility, i.e. alternative machines for each operation, and process plan flexibility, i.e. alternative process plans for each job. Lambert [166] explores the possibility of using process plans captured as AND/OR digraphs for disassembly processes, which are characterized by many degrees of freedom. Applications with complex scheduling requirements have applied the AND/OR digraph formalism to manage the complexity [107]. Due to extensive re-purposing of job-shop scheduling modeling and solution techniques, the disjunctive graph formulation has also been applied to scheduling rail logistics systems [174, 39].

In health care systems, a program of treatments and activities is prescribed for each patient, which includes the type, frequency, and possibly time windows for each type of treatment [208, 241, 142]. While in some cases, patients can be classified and prescribed "a

specific clinical or medical 'pathway', i.e., a standardized process plan for a given diagnosis" [142], often the process plan must be customized to the unique requirements of the patient.

In logistics systems, the classical vehicle routing problem specifies a set of load/unload, or pickup/delivery, tasks for a particular vehicle or fleet of vehicles; see, e.g. disaster relief [123, 22, 294], capacitated vehicle routing problem (CVRP) [54, 277], or warehouse order picking [115]. In disaster relief, logistics processes consist of distributing medical supplies, shelters, sanitation, etc., and casualty transportation of wounded people to medical centers [95, 47]. Process flow is also important in warehouse design, where products arriving at a warehouse are taken through a number processes, including receiving, storage, picking, sorting, consolidation, shipping, etc. [229].

It may not be entirely clear how a vehicle routing problem can be formulated and specified using a formal process plan, but the fact that it can be will pay dividends in formulating abstract interfaces to answer control questions about transportation logistics problems. In the example (figure 14), there is a single depot with a single truck that needs to service four customers. If required services are all pick-up or all drop-off, or at least it's assumed that there are no precedence constraints between customers, then the process plan is given by figure 14a. Suppose, however, all required services are unit load or full-truckload, This requires a constraint to specify that a load's drop-off must be sequenced immediately after its pick-up. However, there are no precedences specifying which load gets picked up first. This scenario's process plan is given by figure 14b. Finally, consider the same required pick-up/drop-off services as in the previous example, except they are not unit load but do require to be unloaded in last-in, first-out (LIFO) order. This scenario's process plan is given by figure 14c.

It's clear that explicitly modeling and constructing these process plans becomes increasingly burdensome very quickly. In math and constraint programming solvers alike, the goal is not to enumerate the entire set of feasible process plans but to fathom and generate quality candidates. This is done by eliciting constraints on valid process plans, specified as valid arc placements on the graph representation. Therefore, what is required is a set of rules for constructing valid process plans, i.e. a meta-model of process planning, given

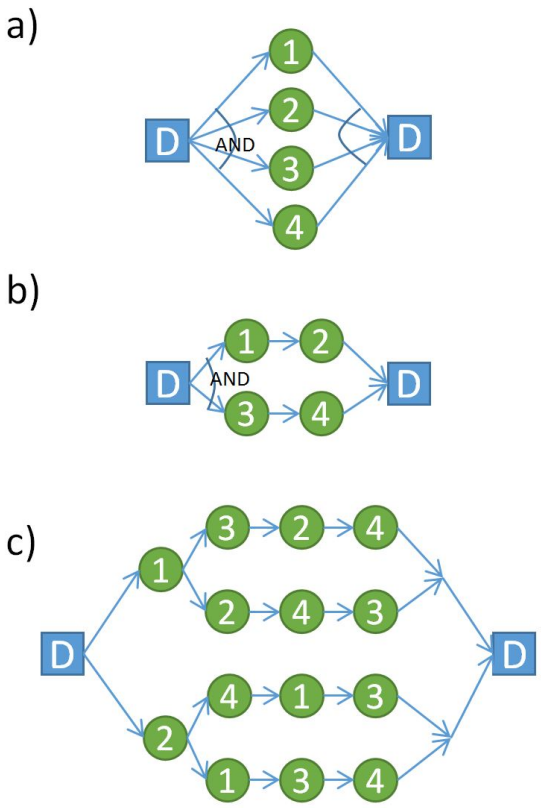


Figure 14: The process plan specification for CVRP where: a) all pick-ups with no sequencing constraints, b) pick-up/drop-off with unit load constraints, and c) LTL pick-up/drop-off with load ordering constraints.

a set of tasks and resources with particular requirements and constraints, respectively. If these rules are modeled as a property of the resources and tasks available to be scheduled, then when new resources or tasks become available to the system, a new scheduling analysis model can be regenerated from the system definition.

2.3 Resource

The DELS owns *Resources* which can execute a set of *Processes* to create a *Product*. Each Resource is defined by the capabilities it possesses, where the capability property (*hasCapability*) expresses the set of processes it can execute. Resource-related research, such as investment or allocation, is among one of the most widely studied topics in industrial engineering; see, e.g., warehousing [229], humanitarian and disaster relief [160], health care logistics [208, 142], transportation logistics [109, 234], manufacturing, etc.

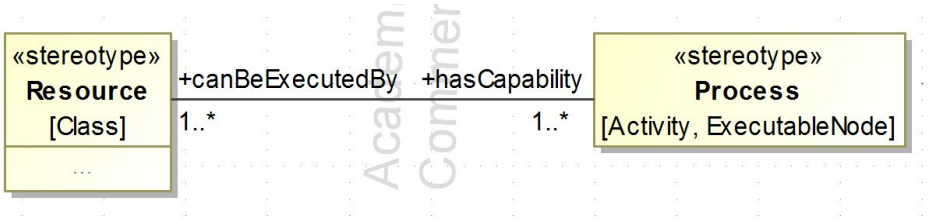


Figure 15: A Resource has a capability set defined by the Processes it can execute.

However, there's a need for precise semantic description of resources because the existing literature gives different names to functionally similar resource types and many authors leave the details of their resources implicit. For example, in some cases *perishable* resources and *renewable* resources are used interchangeably and consider the capacity units as perishable assets, because all unused capacity units before the end of the period have "perished" [132]. The humanitarian, food, and pharmaceutical logistics literature views perishable goods in the more traditional sense when it comes to managing inventory resource levels [301]. *Atomic* resources, which can only perform one operation at a time, are also called disjunctive resources [275] or dedicated resources [126]. On the other hand, *aggregate* or *cumulative* resources can process a limited number of operations simultaneously [275].

A unifying treatment of resource definitions that also makes precise distinction would simplify the modeling of resource problems. For example, Hackman and Leachman [122] separate the inputs and outputs to a process into two classifications, one class for products or materials and a separate class for non-storable services such as labor and machine time. Balakrishnan et al. [18] examine capacity allocation decisions for 'make-to-stock' manufacturing firms that allocate available inventory and 'make-to-order' manufacturing firms that essentially hold production capacity (rather than inventory) in stock. However, when each of these firms experience demands in excess of capacity, each formulates a nearly identical analysis model to allocate the available capacity to different priority customers.

The resource definition discussed in this section is inspired by the OZONE ontology [263], which builds upon the Generic Enterprise Resource Ontology [90] and an early draft of [282], as well as the Dynamic Resource Allocation language in [210]. Zhang et al. [309] proposes an object-oriented manufacturing resource modeling language to encapsulate manufacturing

system knowledge. In addition to the physical resources traditionally considered in logistics models, other classes of resources include information, capital, and organizational resources; see, e.g., [111, 23, 100] for an extensive discussion on the strategic importance of these auxiliary resources. These ontology and taxonomy descriptions have supported the process of abstracting a language element for defining resources in the DELS DSL.

2.3.1 Resource Definition

The attributes and constraints of the resource meta-class define what constitutes a valid resource class (figures 16 & 17). The boolean properties grouped together at the top of the attributes of Resource in figure 16 support constraints that require specific properties or operations to be implemented. For example, if the resource *isCapacitated* then the class must have a set of properties that specify a *capacityMeasure* and *capacityConstraint*. In some cases the boolean attributes only lead to specifying additional boolean attributes, such as whether the capacitatedResource is *isConsumable* or *isReusable*. Then there arise questions such as 'what are the behavioral descriptions of reusable vs consumable resources?'. When the *deallocateCapacity()* method is invoked on a reusable resource, the deallocated resource is returned to the pool and the pool's capacityMeasure is increased.

A taxonomy of resource classes can be constructed using the Resource language and serves as a pattern for using the language for future projects. However, one is not required to sub-class a new resource class from an existing branch of the taxonomy tree, but rather the desired class can be implemented directly from the language. The taxonomy presented here formally captures the OZONE resource ontology [263].

2.3.2 Applications: Capacity for A Capability

Manufacturing capability is based on technical factors of the resources and processes [188]. In flexible manufacturing systems, the tool loading decision defines and restricts each machine's capability to perform specific operations [101], and the capacity for particular processes is measured in terms of total number of tools and the mix of different tool types [106]. When selecting suppliers to form strategic partnerships, Ellram [87] considers the current and future manufacturing capabilities among the most important evaluation criteria. In

Figure 16: The attributes of the Resource language element enable a precise definition of the taxonomy captured in figures 18 & 19.

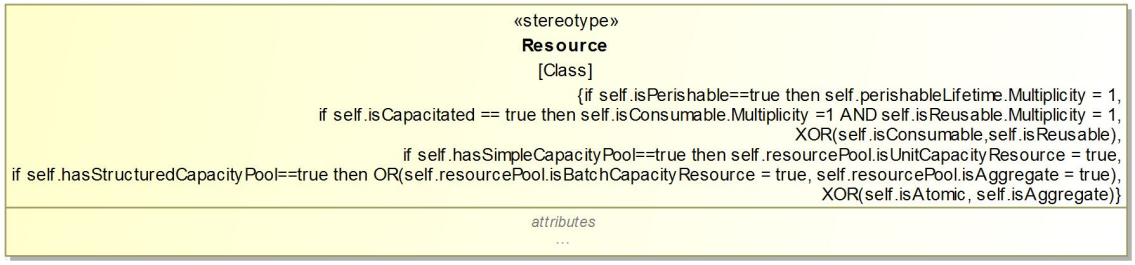


Figure 17: The constraints of the Resource language element enable the checking of correctness and completeness when constructing a class definition.

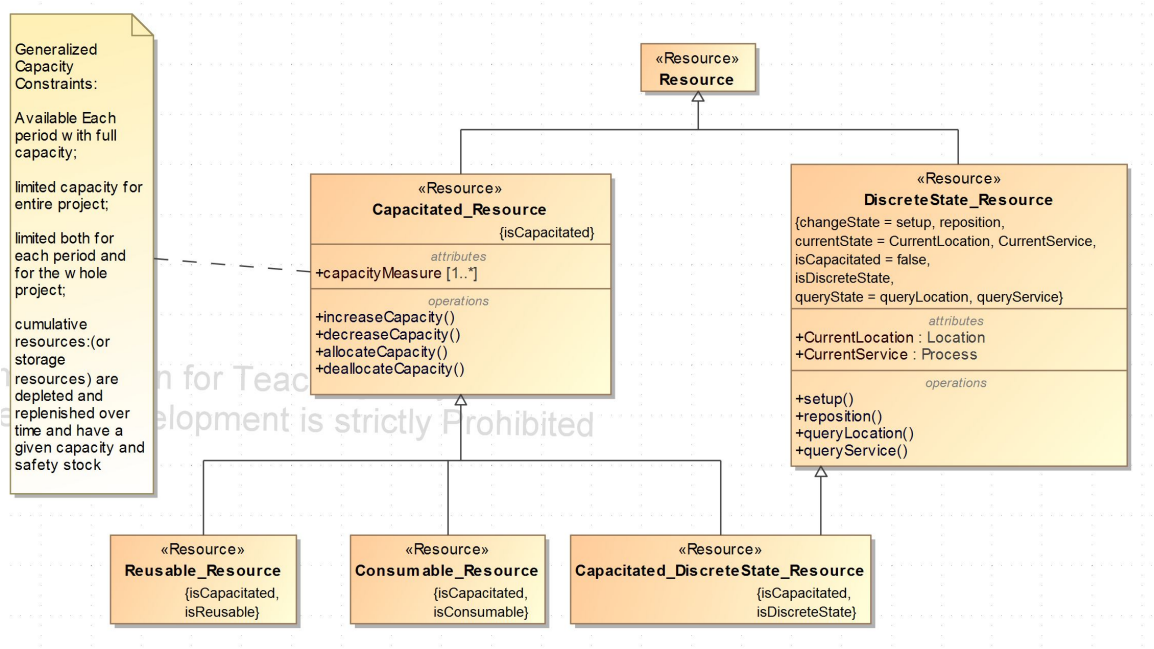


Figure 18: A resource's ability to execute a process can be limited by its state or the amount of its available capacity.

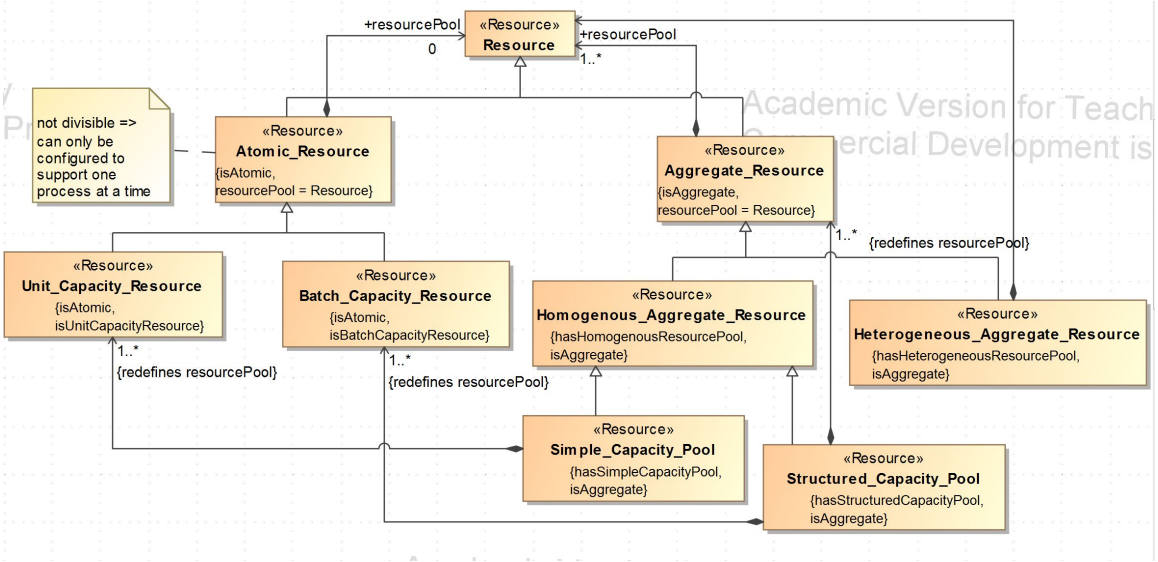


Figure 19: A resource's ability to execute a single process or multiple processes at a time is dictated by whether it is an atomic resource or aggregate resource.

distributed manufacturing paradigms, mechanisms such as brokers try to match capability requests to advertised system capabilities [248, 250].

With *Batch Capacity Atomic Resources* the resource can be used simultaneously by several products requiring the same capability, as long as the total resource capacity used at any time does not exceed the capacity limit. Two examples examples of batch processing: a) 'burn in' operations in the manufacture of circuit boards are performed in ovens that can accommodate several jobs; and b) chemical processes that are performed in tanks or ovens [209]. One interesting nuance encountered when modeling *Aggregate Resources* is that the resource assigned to complete a process may change during processing due to capacity constraints, and this may occur in trucking and health care as well, where the resource (truck or bed) remains assigned, but the driver or nurse changes [213].

In some systems, such as maintenance and repair systems, the resource allocation decision must evaluate the trade-offs between two types of resource capacity, such as allocation of consumable resource capacity and reusable resource capacity; see, e.g. joint allocation of available repair capacity among different items and available inventories to field stocking locations [113, 40].

In health care logistics systems, patients may need to be assigned to specific nurses or doctors having special skills, and staffing decisions require scheduling nurses to fulfill requirements for diverse classes of skills [16, 213, 142]. While applicable to more general classes of resource planning, in disaster relief operations Barbarosoğlu et al. [22] specify the helicopter fleet composition by assigning helicopters from the air force bases to the operation base and the assignment of pilots with given aviation capabilities to the helicopters. In general, different resources and skills are required for different and distinct phases of disaster relief (preparation, immediate response, reconstruction) [160], emergency response in health care [142], or surgical services [208].

In many of the systems discussed in this chapter, a DELS can be described as an *Aggregate Resource* that contains other DELS as members of its resourceSet. These DELS exhibit some degree of independent decision-making and allow the parent DELS to perform several processes simultaneously. This nested structure, or recursive approach, to modeling

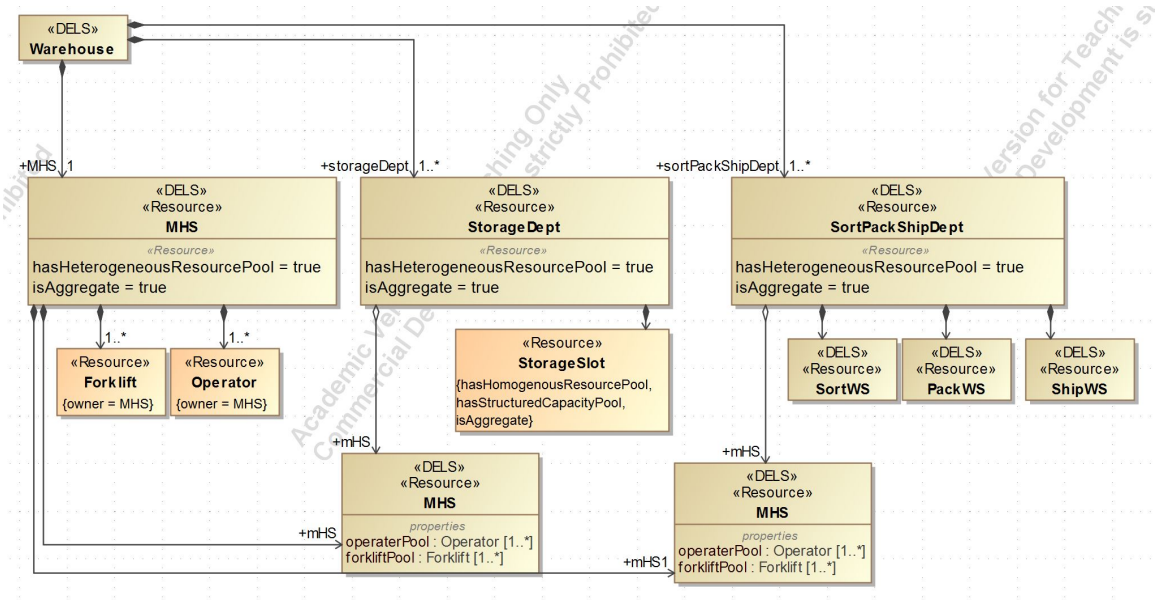


Figure 20: If a DELS nests or owns a sub-network of DELS, then those DELS become part of the resourceSet of the parent DELS.

these systems separates the concepts of decision-making and control (DELS attributes) from capability (Resource attributes), but specifies the exact nature of their interaction (DELS owns resources that define its capabilitySet). This separation of concerns is necessary for constructing a language that isn't pre-disposed or necessarily restricted to implementing any particular control architecture.

In the warehousing model in figure 20, the Warehouse is composed of three subsystems: material handling system (MHS), a storage department, and a sort, pack, and ship department. Each of these sub-systems are DELS themselves and contain resources for carrying out their respective functions. Furthermore, the warehouse's MHS can be modeled as owning all of the disjoint MHS's that belong to the individual departments, but there are other options for modeling the MHS, e.g. allowing the individual MHSs to operate independently and cooperate to complete material handling tasks (an agent-based architecture).

2.4 Facility

Traditionally, the facility is the physical manifestation of the DELS, and its semantic description includes the geometric qualities of the physical space, such as size and layout of resources and spacial relationships among these components. However, a material handling

systems operates within a facility that it doesn't own, and more general transportation logistics systems do not operate within a facility at all. The semantic element of Facility is intended to capture the layout, configuration, and spatial relationship between resource elements in a DELS and can be generalized to include the geometric qualities of the domain as well as geographically dispersed resources and tasks, without necessarily requiring an owned physical space.

Koopmans and Beckmann [158] defined the facility layout problem to configure facilities so as to minimize the cost of transporting materials between them. Drira et al. [82] and Owen and Daskin [199] provide overviews of the facility layout and facility location problems, respectively. Fundamentally, the problem is formulated as $x_{ij} = 1$ if facility i is located at node $j \in N$ on graph $G = (N, E)$ and the costs are based on d_{ij} and f_{ij}, travel distance and flow quantity between nodes i and j on network G. In the CVRP formulation by [77], all the addresses are located on a network $\mathcal{G} = (V, E)$ with a set V of nodes and a set E of (undirected) edges and (directed) arcs. In the description of the material handling system, a set of addressable locations is associated with each material handling device, where an addressable location is a physical location to which a material handling device has access to pick objects up or put objects down [260]. In Core Manufacturing Simulation Data (CMSD), layout information is used to define spatially-oriented characteristics and interrelationships for the logical and physical entities that are used to carry out production activities [225].

While not actually all that different, the difference between these formulations is specified by the *isLogical* attribute of the *Facility*, which describes whether the facility is a physical artifact of the system, as it is in manufacturing plants, or if it merely a logical construct for organizing the spatial and flow relationships between entities in the DELS. The *hasGeographicLayout* could easily be thought of as a *hasSpatial* property. Additionally, in many problems the physical artifact is irrelevant to the analysis or specification, but a constraint is enforced that if the physical artifact is presumed to exist, then the geometric layout must be specified, i.e. *hasGeometricLayout == true*. Otherwise if *hasGeographicLayout==true* and *hasGeometricLayout == false*, then the entities resident in the DELS

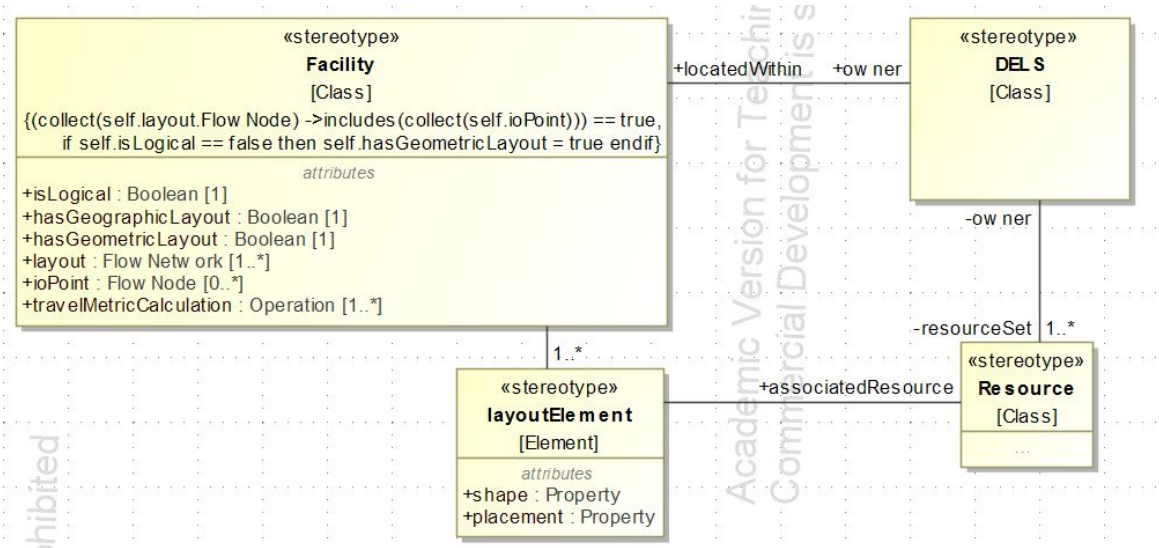


Figure 21: The Facility maintains the layout and address data of all the resources and customers within the scope of the DELS.

may be thought of as massless points, or lines in the case of conveyors. This is a particularly convenient 'exception' for analysis driven use cases. Both of the Geometric and Geographic layouts can be constructed using *layoutElement*, which specifies the location, footprint, and orientation of each resource within the facility. The *layoutElement* integrates with CMSD specification, which provides a far more extensive language for modeling the layout of a manufacturing facility [225]. The input/output, or interface point, to each resource is associated with a node, or address, on a graph description the Facility's *layout*. In analysis models, the facility can use its layout, a *Flow Network* structure, to provide feasible paths between two resources on the network.

2.5 Interoperability and Integration: An Application of the DELS DSL

Formal domain modeling methods, including the creation and usage of domain-specific languages, reference architectures, and model-to-model transformations, are a key enabler to achieving integration and interoperability across system models and analysis tools in the DELS domain. A domain-specific language (DSL) provides a common language to formulate and solve problems in a particular domain, and the remainder of this section will discuss applying DSLs to solve interoperability and integration challenges encountered in modeling and analysis of DELS.

2.5.1 Interoperability Between System Models and Analysis Models

"To achieve interoperability using [a DSL], semantic mapping is needed for various reasons. The same term may have different meanings in different applications and universes of discourse" [52]. Incomplete and implicit system descriptions and incompatible semantics limit the ability to reuse and apply analysis models and solution methods to semantically similar domains; see, e.g. the application of machine scheduling algorithms to train scheduling [198, 174, 39], surgical case scheduling [208], and warehouse dock scheduling [115], among many others. In these cases, the process of deciding whether two systems are similar enough to apply an existing analysis model and solution algorithm from one system to another is less than obvious, and often the semantic relationship between the two systems is ambiguous; e.g. is train scheduling a sub-class of job shop scheduling?

A DSL, and its supporting implementation patterns, provide a partial solution to this interoperability challenge by providing a common language to specify both the system and analysis models. The DELS DSL discussed throughout this chapter provides the additional benefit of abstraction, such that system models and analysis models from seemingly different domains, such as health care and manufacturing, both can be formulated using product, process, resource, and facility (PPRF) semantics. The result is a consistent and, due to well-formedness rules, possibly complete representation of the two systems and their associated analysis models. Then the semantic relationship between the two systems should be obvious and the application of one analysis model to a system model, or domain, for which it was not originally intended should be a straightforward process.

Proper application of the DSL to model both system and analysis models has the promise of creating plug-and-play solution algorithms for DELS. While an exciting prospect on its own, this is a critical requirement for a controller to be configured with or have access to a family algorithms to solve a wide variety of control problems. In the remainder of this section, a use case will demonstrate the usage of a DSL to bridge between a system model and an analysis model with the goal of applying the target solution method to an instance of the system model.

Resource planning and control problems are among the most well studied operations

research problems, and since the operational control of resources will not be formally addressed until chapter 3, the interoperability use case focuses on a resource investment and planning problem. This class of analysis problems provide a method to determine the optimal capacity levels for each functional capability, which is typically provided as a given in the statement of the operational control problem. In this example, a job shop example [110] has been selected as the source system model, while a resource investment and planning model for capacitated processing networks [288] is the target analysis model and is assumed to have an existing tool implementation whose reuse is the goal of the demonstration.

By mapping the job shop system to PPRF semantics, the existing resource investment analysis model, which has been mapped to the same PPRF semantics, can be applied to answer the desired question about the job shop. The goal is to show that any system that conforms to the DELS language, either through usage of the language or a mapping to the semantics, can take advantage of any analysis model that also conforms (or can be mapped) to the language (Figure 22). A complementary view of this process is that the PPRF serves as a bridging abstraction between system models and analysis models.

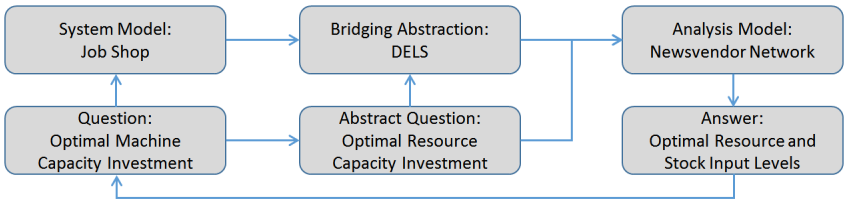


Figure 22: An Overview of the Mapping between the System Model to Bridging Abstraction to Analysis Model.

In this (loosely re-purposed) job shop scheduling problem [110], suppose that n jobs have to be processed on m machines (Figure 23lhs). In the associated scheduling problem, the system has a given number of machines with processing capacities for each operation; therefore the capacity is a pre-defined given. Since the scheduling decision will be addressed later in this dissertation, this example is currently interested in exploring how the aggregate production planning and strategic resource investment decisions are made; i.e. "how much [processing] capacity for each capability (operation) should the job shop have to meet the required throughput of jobs?". Additionally, raw material constraints are an added

consideration for this use case.

For this example the analysis model used will be a Newsvendor Network. Newsvendor Networks are a broad class of capacitated processing networks that incorporate the effects of capacities, inventory, and discretionary (or optional) activities [288]. In this analysis model, each *Activity* requires a fixed amount of capacity from a set of *resources* and *input stocks* (Figure 23rhs). The activity is executed to satisfy *demand* requirements. Formulated as a stochastic optimization with recourse, the analysis is divided into an ex-ante decision on the resource and input stock levels and an ex-post (after demand realization) decision on which activities to execute to satisfy demand.

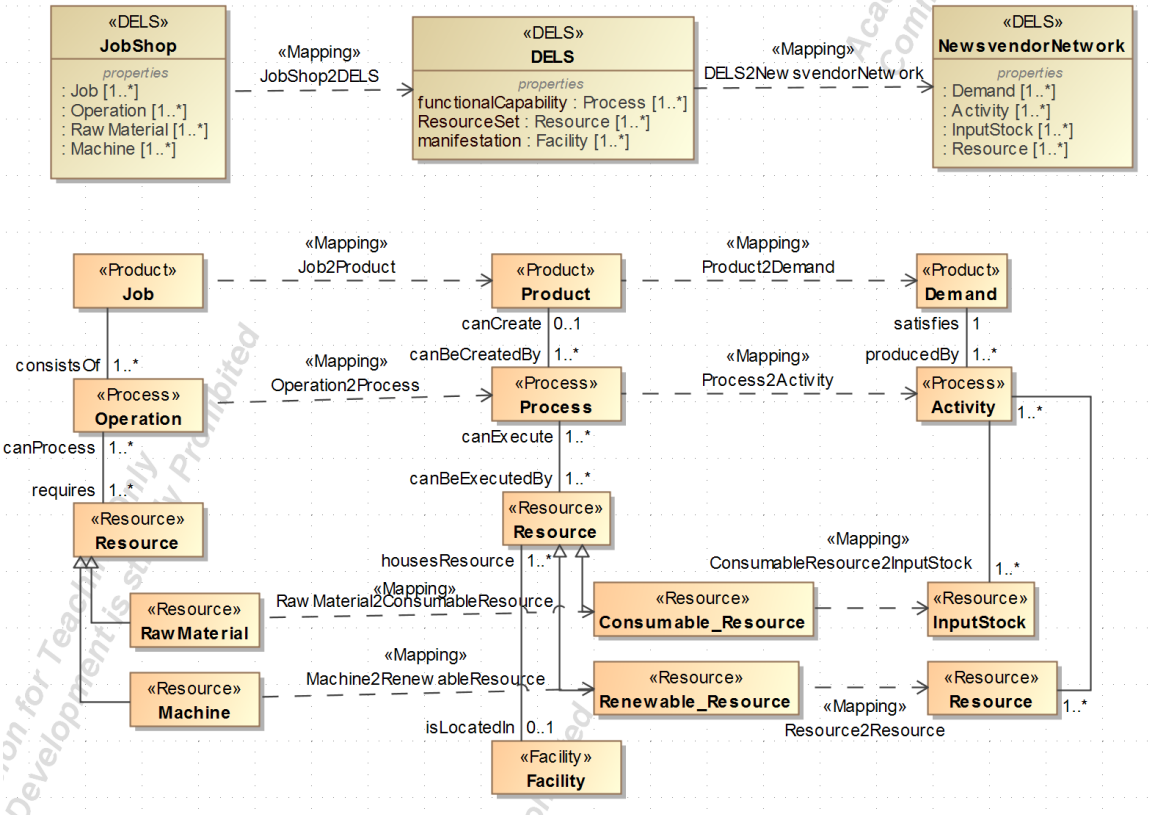


Figure 23: The SysML Block Definition Diagram of the Mapping between the System Model to Bridging Abstraction to Analysis Model.

First, the predefined analysis model for resource investment and planning in capacitated processing networks, the Newsvendor Network model, is described using, or mapped into, PPRF semantics (figure 24). This encapsulation process makes the existing analysis model and its corresponding tool compatible with the system model, and therefore accessible and

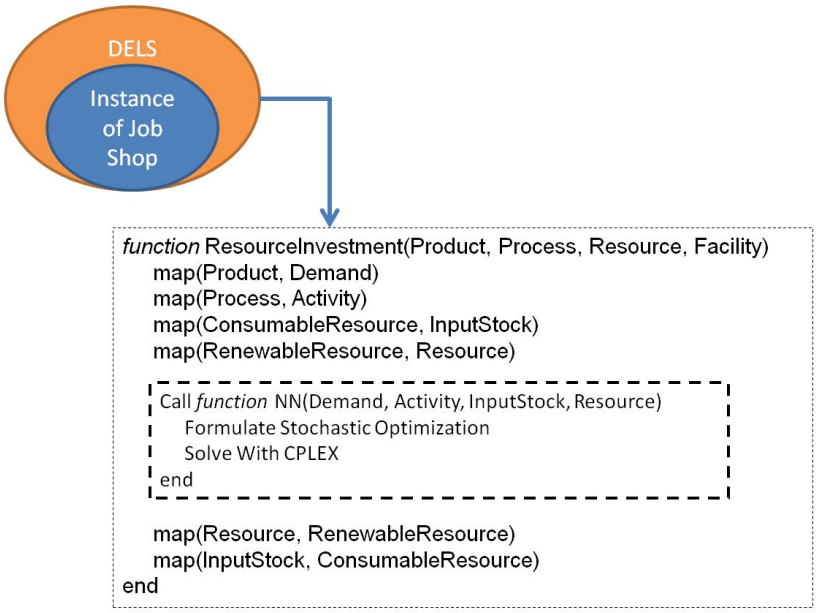


Figure 24: The existing analysis model is mapped to the DELS semantics and encapsulated in a reusable analysis object.

reusable across a broad range of system models that conform to the PPRF semantics.

Then in a similar fashion, the job shop scheduling problem is mapped to PPRF semantics. This mapping can be done deliberately, by constructing the system and analysis models using the DELS DSL, or can be done ex-post by applying the stereotypes to the existing models and cleaning up the semantics as necessary. In either case, the result is semantically compatible elements in both the source system model and the target analysis model.

By expressing the models in a common language, the predefined analysis model that conforms to one system description can be applied to a system model that conforms to a different system description. That is, given an instance of a job shop system model and the associated machine capacity and raw materials investment question, a transformation can be executed to produce an instance of the newsvendor network analysis model, which can be used to answer the associated resource investment problem (figure 22).

While applying this method to encapsulate and reuse existing analysis methods and tools is conceptually straightforward, often it is difficult to precisely interpret the semantics that an author has used to describe a particular system or analysis model. For example, it

would be more precise to state that the job authorizes or requires the execution of a process to produce a product, rather than representing the job itself. In this context however, no product is explicitly described so it is assumed that the job itself is flowing and is the product. Often the exact nature of relationships between semantic objects is implicit, which causes difficulty in reusing these semantics and more importantly, applying the solution method to answer the same question in a different instance of the system.

Additionally, ontologically incomplete system and analysis models inhibit the direct reusability of existing models:

> "The perception that ontology can solve interoperability problem by mapping terms between different systems is very evident. This view is largely founded on an assumption that the underlying conceptualizations that these systems comply to are equal and the only differences are those related to terms which are used to refer to different concepts. If this would really be the case then the solution to interoperability would be somewhat tedious but relatively simple. Unfortunately this is not the case since the underlying conceptualizations of those systems can be represented with conflicting paradigms with models which are not ontologically sound in the first place." [114]

This is a fundamental issue that underscores the need for improved modeling methodologies for the DELS domain. While much of the existing literature must be interpreted from the source material, applying model-based methodologies, including the application of the domain-specific language discussed in this chapter, has the potential to improve the interoperability of system and analysis models across the DELS domain.

2.5.2 Integration of Systems Models

In traditional supply chain logistics systems, the manufacturing, warehousing, and transportation systems are distinct subsystems which manage their own goals and resources. Coordination between these systems is difficult and often executed in an asynchronous manner due to the information and control systems being isolated from one another (figure 25). However in next generation logistics systems, improved information sharing and

system integration has the potential to dissolve the silos that traditionally segregate these domains. For example, material handling systems linking many diverse systems will be able to communicate and coordinate the movement of a product from a warehouse to its needed location on a manufacturing line. Manufacturing lines, inventory systems, and transportation systems will coordinate the completion of orders, possibly consisting of a mix of products still to be manufactured and those already in storage, with distribution resources to meet customers demands. A common reference model and language for describing these systems and the problems they need to solve cooperatively is absolutely critical to achieving next generation smart logistics.

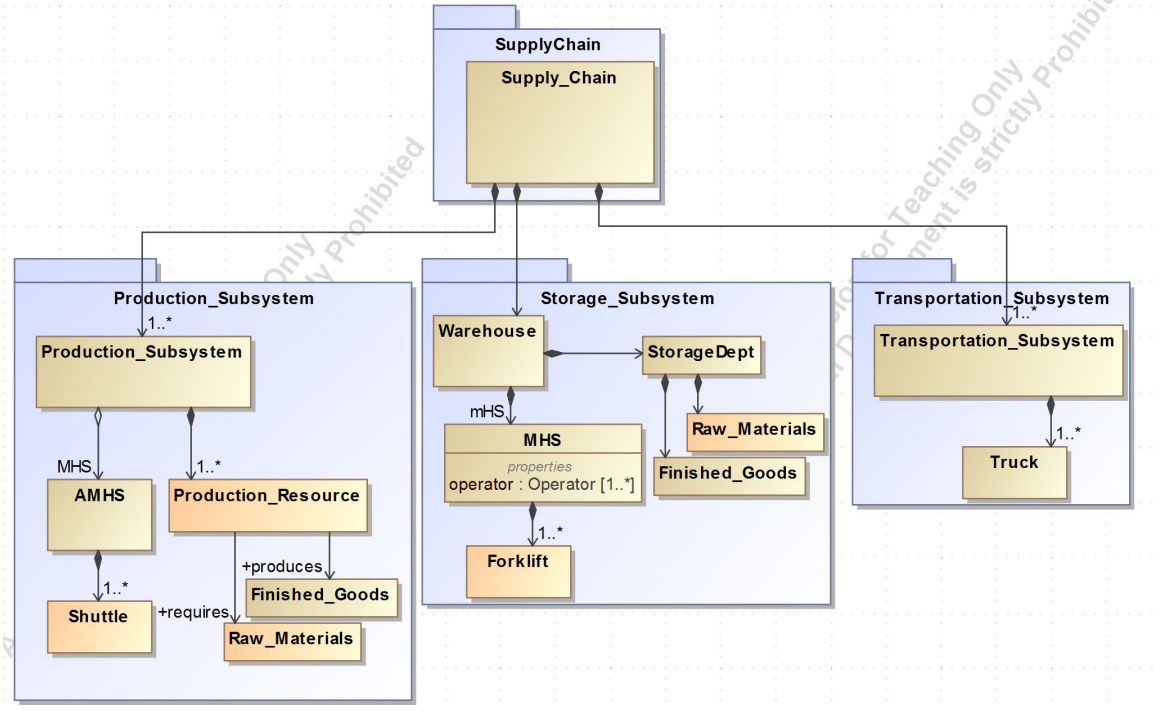


Figure 25: In traditional supply chain systems, the production, storage, and transporation functions are isolated.

In figure 25, the production, storage, and transportation systems are modeled independently as loosely related sub-systems of the supply chain. In the broader supply chain context, it may be understood that the transportation subsystem, the warehouse MHS, and production AMHS are all jointly responsible for moving materials around the supply chain, but in practice this process is controlled by three disjoint systems, the TMS, WMS, and

MES respectively. Furthermore, it's unclear whether the finished goods and raw materials in the storage department are the same as those used by the production system; once again, two separate information systems may track these materials independently.

In figure 26, the PPRF semantics have been applied to the model to clarify the resources and DELS being modeled and to refine their relationships. Each of the independent material handling systems have been consolidated into a system-wide MHS that views and coordinates the independent MHSs as if they belonged to its resource set. However in the previous figure, it would have been difficult to merge these systems due to lack of precision describing the system's resources; e.g. Are the AMHS and transportation subsystem's Trucks equivalent classes? Furthermore, the storage department and production resource now carefully define a joint set of raw materials and finished goods.

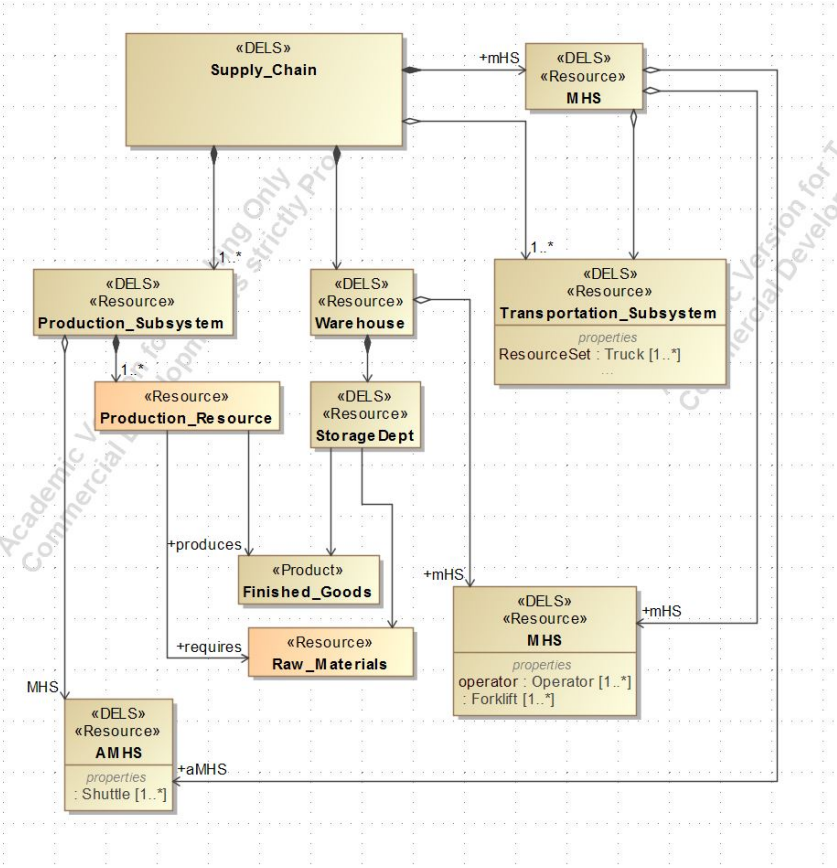


Figure 26: A common language for describing DELS supports the integration of traditionally isolated systems.

2.6 Summary

A domain specific language for DELS that formalizes the product, process, resource, and facility (PPRF) pattern that is common throughout system models in this domain, establishes a precise language to express system and analysis models. This common language is integral to providing interoperability between system and analysis models and integration between different systems' models, and provides a pathway to supporting a uniform interface and plug-and-play access to analysis models and their corresponding solution algorithms and tools to support a wide variety of control questions across the DELS domain. This is a critical requirement to providing real-time, on-line decision making support for operational control for DELS.

CHAPTER III

CONTROL QUESTIONS IN THE DELS DOMAIN

Designing the operational control mechanisms for discrete event logistics systems first requires a description of the control activities that are being executed by the system controller at the operational management level. A standard description of the control problems in the DELS domain remains a challenge because the names of the control problems may vary by author or problem class leading to overloaded, under-defined, and incomplete specifications; see, e.g [92, 25, 136, 133, 41, 108, 270, 197]. One strategy to overcome this challenge and make the extant research more portable is to strip away the domain-specific semantic content from the problem and operate on the abstract mathematical formulation, i.e. establishing syntactical interoperability [85]. However, the mathematical abstraction explicitly ignores and is not capable of capturing the behavioral aspects of the system. Therefore developing a controller that can access a library of plug-and-play analysis tools to solve a wide variety of control problems requires developing semantic interoperability between system and analysis models. The DELS language discussed in the previous chapter provides the necessary semantics for bridging between abstractions of the behavior of the system, but there remains the challenge to capture a specification of the control activities.

To design the controller and its implementation mechanisms, the reference architecture must capture explicitly the complete set of control problems that the controller must be able to solve in order to be effective in managing the behavior of the system. In this section, it is argued that there is a fundamental set of control questions which can be extracted from a relatively small collection of distinct control problems that are addressed in the literature; e.g., the theory of controllable queueing systems specifies the control of admission, servicing, scheduling, and routing jobs in queues and networks [270]. To construct a canonical abstraction that is independent of the specific context of the problem, the formulation presented here instead relies on the fundamental research questions being

addressed in the literature (figure 27), and questions that can be derived from observing the flow of a task through the system and its interactions with the resources. These questions are: (1) 'Should a task be served?'; (2) if so, then 'when should the task be serviced?'; and, (3) 'by which resource?'; (4) finally, 'where should the task be sent after it's complete?'; (5) as well as the resource-related 'when does the state of a resource need to be changed?'.

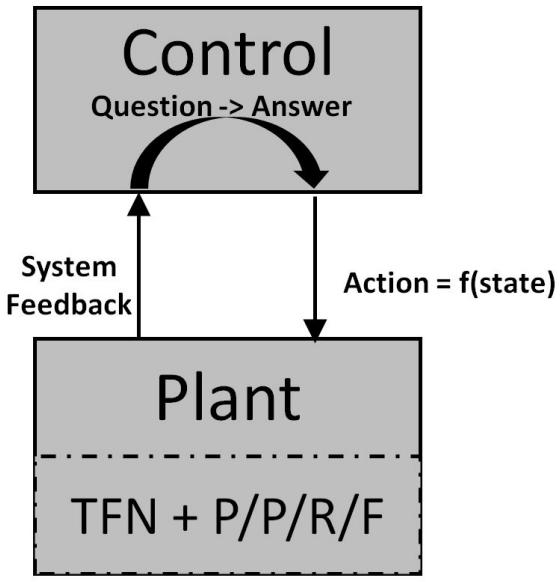


Figure 27: A canonical set of control questions define a comprehensive functional specification of all the decision-making mechanisms that a controller needs to provide.

This fundamental functional specification of DELS control behavior is formalized as equivalence classes of control questions defined by a standard abstraction formulated using the abstract PPRF semantics discussed in the previous chapter. A standard representation of each class of problems enables a uniform interface to solution tools thereby creating opportunities for interoperable, or plug-and-play, analysis tools. Therefore, equivalence among the members of the class is defined by a common abstract interface definition, where two problems are considered equivalent if they share the same fundamental input/method/output definition. Furthermore, the question-driven organization enables each control problems to be tied to a functional mechanism, or specific behavior, of the system. These functional mechanisms will be discussed in section 4.5 and the interface will be discussed in section 4.3.

Each of the following sections presents a textual statement of the control question using the DELS language, as well as a semantic-free, or neutral, mathematical formulation that is required by the analysis models. The applicability of this abstract definition to concrete domain-specific analysis models is discussed. A summary of the relevant literature is given in table 1. The rest of this chapter is organized around the questions themselves: 'which tasks to service?' (section 3.1), 'when to service a task?' (section 3.2), 'which resource is assigned to service a task?' (section 3.3), 'where to send a task after it's complete?' (section 3.4), and 'when to change the state of a resource?' (section 3.5).

Table 1: Overview of Literature on Control Questions in DELS.

Domain	Admission	Sequencing	Resource Assignment	Routing	Resource State
Manufacturing	[156, 132]	[4, 110, 31, 127, 3, 284, 298, 63, 34, 139, 49, 227, 51]	[58, 298, 170, 304, 44, 139]	[238, 25, 298, 170, 304, 249, 36, 101, 201]	[156, 270, 209, 89, 112, 272, 134, 162]
Warehouse		[115, 226, 67]	[229, 115, 226, 211, 30]	[207, 115, 67]	[229, 203, 226]
Transportation	[77]	[92, 240, 77, 75, 83, 135, 169]	[109, 234, 306]	[168]	[77, 108, 300, 70, 60, 10]
Inventory	[276, 217, 18, 120, 46, 125, 98, 88, 151, 179, 8]	[92, 41, 169, 153, 49, 48, 76, 51]	[18, 288, 40, 14]		[149, 68, 205, 171]
Health Care	[164, 293, 142]	[178, 78, 208, 142]	[178, 28, 208, 213, 142]	[142]	[142]
Humanitarian		[123, 22]	[37, 123, 95, 200, 20, 222, 289]	[22]	[221, 289, 22]
Maintenance			[113, 112, 40, 194, 265]		[105, 43, 194, 43, 265, 247]
General Models		[161]	[103, 192, 133, 210, 288, 61, 21]	[180]	[176, 86, 132, 291]

3.1 Problems that deal with which tasks to serve

When demands exceed of capacity of the system's resources, a controller is presented with the problem of allocating the available capacity to different classes of products, or customers, resulting in the rejection of some requests from lower priority demand classes or from less valuable products in anticipation of future demand from the more valuable or higher priority classes [18, 120]. More generally, a controller must decide if it wants to service a particular task when it becomes available to the system. However rejecting a task, either explicitly by denying it entrance to the system or implicitly by delaying service long enough for the customer to balk or renege, typically incurs a cost to the system. Deciding which tasks to service creates the ability to control the rate of arrivals by allowing or rejecting tasks [156]. The development of good acceptance rules, coupled with capacity-planning tools, leads to greater control over the use of resource capacity and increased profits by improving decision-making such as due-date quotation, price quotation, and contracting additional capacity [132].

The decision to reject tasks is often dependent on the state of the system, such as maintaining system stability with short service times, but deciding which tasks to reject is managed by a policy that must balance a collection of factors. Back-ordering is a flexible strategy that complements a pure admission or rejection decision where lower-margin orders might be backlogged before inventory is depleted in order to reserve units for possible orders from high-margin customers [46]. Rabinowitz et al. [217] propose a partial back-ordering control policy that rejects customers when the back-order queue is greater than a certain threshold. In some variants of the CVRP, only a given subset of customers must be visited and the others may be visited if it is profitable to do so [77].

Admission control, such as advanced access or walk-in policies, plays a significant role in managing waiting times for health care systems [142]. Patients can be classified as elective, urgent, or emergency, and admission planning in general hospitals means selecting elective patients from a waiting list or determining the optimal mix of patients to be admitted in order to obtain optimal utilization of the available resources while reserving capacity for emergency admissions [164, 293].

What determines the less valuable class of demand? Market segmentation, demand shaping, and pure pricing measures are prominent strategies for establishing demand classes. A simple motivating example based on differing marginal profits is given in [276]. In this example a company has a national price list for its product, but transportation costs are different to different parts of the country, which results in a profit per unit sold that is a function of the geographic origin of the demand. In an assemble-to-order system with limited inventory of a common component, the controller must ration the common component to its (two or more) end products, which have different profit margins [18]. However, is there a more general model of setting prices and accepting tasks for service?

Revenue management involves the use of pricing and inventory control strategies to balance supply and demand in a revenue-maximizing manner. Firms must manage both sales (demand rationing) and pricing decisions in order to maximize profits. Keskinocak and Tayur [151] discuss order acceptance in the context of due-date management, including pricing decisions. Harris and Raviv [125, 88] review and discuss pricing schemes that are commonly used in the marketing of many different products, including offering a single price, auctions, and priority pricing. Chen [50] considers a problem inspired by e-commerce where a firm offers a menu of price and lead time combinations, and customers can choose their priorities.

The parameter selection mechanism may be modeled or implemented as a sequential decision problem where "firms may initially select their waiting-time standards, followed by a selection of their prices in a second stage (service-level first). Alternatively, the sequence of strategic choices may be reversed (price first) or, as a third alternative, the firms may make their choices simultaneously (simultaneous competition)" [8]. In the project management literature, the order arrival process includes a request for proposal, which must be responded to by offering a price, lead time, and any other contract parameters for the task [132]. This processes addresses how the cost, penalty, and performance parameters are determined, where in many other papers, these parameters are assumed to be given. This behavior closely aligns with the mechanics of the contract net protocol used in agent-based and holonic architectures (see section 4.1.2), and in that sense, this helps us move towards a

complete specification of the problem at hand.

Maglaras and Meissner [179] provide a unifying framework for the "dynamic pricing" problem, where the firm influences the demand for each product by varying its price and the problem is to choose a dynamic pricing strategy for each of its products, and the "capacity control" problem, where the firm selects a dynamic capacity allocation rule that controls when to accept new requests.

3.1.1 Formulation

In many of these formulations the arrivals are static, i.e. all tasks are available at time zero, or it is assumed that the tasks arrive each period in batches . $\hat{\mathcal{L}}_t$ is the set of tasks that first become available in time period t and $\mathcal{L}_t$ is the set of tasks available at time t before new arrivals are added to the system. Then $\mathcal{L}_t^+ = \hat{\mathcal{L}}_t \bigcup \mathcal{L}_t$ is the set of tasks available to service at time t [108]. Suppose the tasks needed to be explicitly accepted first before being added to the system, then this formulation is extended to selecting $\mathcal{L}_t^* \subseteq \hat{\mathcal{L}}_t$,$\mathcal{L}_t^+ = \mathcal{L}_t^* \bigcup \mathcal{L}_t$ is the set of tasks available to service at time t, and $\hat{\mathcal{L}}_t \setminus \mathcal{L}_t^*$ are the tasks rejected from the system at time t. Additionally, this could be formulated to support dynamic selection problems, where tasks arrive one at a time. In this extension, $y_i = 1$ if task $\ell_i \in \hat{\mathcal{L}}_t$ is accepted [259].

In posted-price formulations, the price schedule for a single product, $\{(p_k, \tau_k)\}_{k=0}^K$, where K is the number of segmentations and p_k is the offered price if a customer agrees to lead time τ_k [50]. Then $\lambda(p, \tau)$ is the aggregate demand as a function of the quoted price and lead-time [98, 50, 206]. In addition to using price to shape the demand curve, Maglaras and Meissner [179] also consider a capacity control variable, $u_i(t)$, which is the probability of accepting a request for product i at time t, and has an equivalent effect on demand shaping as the price setting mechanism.

3.2 Problems that deal with when a task in system gets serviced

Once a controller decides that a particular task will be serviced, it must then decide when it will be serviced. While a sequence gives the order in which jobs are to be done, a schedule can be expressed as a fixed or dynamic sequence of tasks to service, with implicit or explicit expected start times; therefore, a sequence is a simplified schedule [139]. A predictive

scheduler would specify an entire sequence of tasks in advance, whereas a reactive scheduler would specify each task (or subsequence of tasks) in real time [284]. Adaptive policies, such as dispatching rules, specify the next task to be processed and produce an implicit sequencing of the tasks awaiting service [31, 298]. Hausman and Scudder [127] examine the performance of dynamic dispatch rules which use inventory status information and work-in-process inventory information and outperform dynamic rules that do not incorporate those data.

3.2.1 Formulation

The sequencing decision may produce a continuum of solutions that range from establishing a complete sequence of all the tasks in the system to only a partial sequence of some subset, which may include determining only the next task to be serviced.

A complete sequence of all tasks in the can be expressed as $\sigma = \{l_1, l_2, \ldots, l_n\},\ \forall l_i \in \mathcal{L}_t^+$. Several types of partial sequences of the tasks can be expressed by partitioning the set of tasks into K partitions: $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_K\}$, where $\sigma_k = \{l_{k(1)}, l_{k(2)}, \ldots, l_{k(n_k)}\}$ is the sequence of tasks in the k^{th} partition [66]. For example, if the control decision assigns a task to a particular time bucket, then the partitions correspond to a temporal partitioning of the decision space. Furthermore, the formulation can be reduced to two time buckets: this period and all other time periods. Then the controller must only decide if a particular task is going to be serviced in this period or not. If the resource assignment decision has already been made, then the partitions can correspond to the assignment of tasks to each resource: $\sigma = (\sigma_1, \ldots, \sigma_k, \ldots, \sigma_K)$, where σ_k is a sequence of tasks on resource k. The sequence σ_k on resource k can be represented by a set of ordered tasks, $\sigma_k = \{l_{k(1)}, l_{k(2)}, \ldots, l_{k(i)}, \ldots, l_{k(n_k)}\}$.

The variable X_{lt} may denote that task $l \in \mathcal{L}_t^+$ is serviced in period t [108]. More generally, X_{lk} may denote that task $l \in \mathcal{L}_t^+$ is assigned to partition k, and $X_{lk(i)}$ may denote that task $l \in \mathcal{L}_t^+$ is sequenced in the i^{th} position within partition k.

3.2.2 Applications

The job sequencing problem is well studied in the manufacturing domain resulting in many early results in complexity theory and heuristics, which have been broadly applied to adjacent domains; see, e.g. [4] for an early collection of single machine, job sequencing problems and solution methods, and [110] for an overview of job sequencing problems in more general manufacturing system. For a more modern treatment of a rather new problem, Boysen et al. [34] provide an overview of sequencing problems and models in mixed-model assembly lines, where different products can be jointly manufactured in intermixed product sequences (lot size of one) on the same line. Using a disjunctive graph formulation of the machine scheduling problem, [63] implement resequencing (and machine assignment) methods by moving connective arcs around in the graph. The shifting bottleneck heuristic exploits the separability of the sequencing decisions and sequences jobs on machines sequentially [3].

In vehicle routing problems, the sequencing decision determines the order in which the tasks are serviced [92]. Additional extensions to the CVRP problem explicitly state a required time window for the pickup or drop-off [240], therefore attaching a specific service time to the task, rather than an implied one. In periodic VRP, a two-stage sequencing problem first assigns the task to a particular partition of the service cycle, and then provides a sequence to all the tasks assigned to a particular partition [77]. In warehouse AS/R systems, both storage and retrieval requests can be sequenced on the same resource, known as dual command cycles [226]; this problem is similar to the sequencing of pick-ups and drop-offs in the CVRP (CVRPPD) [75]. These methods are used in the disaster relief domain to sequence dispersion of relief supply resources [123].

In the health care scheduling literature, patients are divided into two or more classes, including a class for emergency cases that require immediate treatment, and the control problem is to sequence the patients in the waiting room while accommodating existing appointments [178, 208, 142]. While Pham and Klinkert [208] use a job shop formulation to sequence surgical cases, Dexter et al. [78] consider several policies: minimization of the average waiting time (for both patients and doctors), (2) FCFS basis, and (3) medical priority.

The frequent reuse of job shop and vehicle routing sequencing heuristics to adjacent domains suggests an underlying commonality to the sequencing decisions made across the DELS domain. The following sections discuss a collection of common sequencing decision-making problems that can be formulated using the formulation given in section 3.2.1, including coordinating multiple tasks, delaying service of a task, batching tasks, and splitting a task.

3.2.2.1 Coordination of Tasks

Sometimes the sequencing, or timing, of a collection of tasks is arranged to coordinate a collection of resources. A number of authors have considered an enriched version of the vehicle routing problem in which both the timing and amount of deliveries (as well as the schedule of the vehicle fleet) must be determined so as to ensure that customers do not run out of inventory [92, 41]. The joint replenishment problem is thoroughly reviewed in [153]. Vendor managed inventory (VMI) arrangements allow the vendor to determine the optimal timing and quantity of replenishment to the retailers [92, 169]. Lee et al. [169] consider the integrated timing effects of shipping consolidation and inventory replenishment decisions in a VMI environment. Chandra and Fisher [49] and Chen [51] investigate the value of coordinating production and distribution scheduling, and more generally, the timing of tasks to coordinate the usage of different resources and subsystems.

3.2.2.2 Delay Service of a Task

Additionally, the control decision can simply be to delay service of a task until a later period. In backordering and inventory rationing models, Deshpande et al. [76] propose a model where orders can be backordered even when stock is available in order to reserve capacity for higher priority customer classes. In Çetinkaya and Lee [48], the supplier has the option of delaying delivery in anticipation of orders from other retailers. Cattani and Souza [46] demonstrate that direct channel firms can postpone a shipment to a customer until later and then use a more expensive and faster transportation means with the objective to reduce demand uncertainty in the mean time and improve customer satisfaction (service level). In response to unplanned events such as emergencies or other disturbances

to the planned appointment schedule, health care systems can respond by dynamically (re)scheduling patients (to future appointment slots) to improve patient waiting times and resource utilization [178, 142].

Batching and demand splitting are two 'delay service'-type decisions, where batching chooses to delay immediate service to improve the system efficiency and demand splitting delays the completion of part of a task until a future period to meet constraints in the current period.

3.2.2.3 Batching Tasks

Also known as lotsizing, batching decisions cluster items for transportation or manufacturing processing at the same time [161]. Batching delays the service of some jobs, in order to more efficiently process them (use resources more efficiently through reduced setups). However whereas an online sequencing algorithm has the flexibility to consecutively sequence any number of similar jobs, batching is a more rigid policy that waits for a specific predetermined number of similar jobs.

In multi-item inventory systems, economies of scale can be realized by coordinating replenishment orders for groups of items [93], which also applies when there are several locations instead of several products: e.g., a central depot coordinates the replenishment process for a set of locations with demand for a single commodity. In the warehouse order partitioning problem, orders must be partitioned in time for assignment to waves and pickers [115, 226, 67]. In freight consolidation, for each arriving order the system must decide whether to ship it immediately by itself, ship it immediately as a part of a consolidated load (the order is the release trigger), or delay shipping to consolidate the order into a future load [135]. In manufacturing production scheduling, just-in-time shipments can be partitioned into some number of production batches [227].

3.2.2.4 Splitting a Task

When permitted, demand or lot splitting decisions determine when a task should be split to accommodate capacity constraints, due date constraints, priority constraints, etc. Often activity splitting and preemption are considered to be interchangeable [53], though there

may be varying preemption penalties including time-delays, such as restarting a job from the beginning, or setup costs.

The split delivery vehicle routing problem (SDVRP) allows the demand of a customer to be satisfied by more than one delivery. Dror and Trudeau [83] propose heuristics for the split delivery VRP where routes are broken and re-combined to improve the solutions quality, both on total travel distance and the number of vehicles required. Barbarosoğlu et al. [22] use this model for the distribution of disaster relief supplies where "partial service to the nodes [is] allowed, so that the demand of any node can be satisfied with different helicopters during multiple number of visits."

At first inspection, each of these applications seem very different from one another, but by examining the fundamental structure of the control decision, it becomes clear that each of the control applications discussed in this section belong to the same equivalence class of control problems, i.e. conform to the same formulation given in section 3.2.1. A uniform or common formulation for a class of control problems is a partial solution to organizing and creating interoperability for the corresponding class of analysis tools.

3.3 Problems that deal with which resource is assigned to service a task

Broadly speaking, the assignment problem consists of assigning tasks to be fulfilled by a particular resource, thereby allocating a portion of the resource's capacity to servicing the task. These types of problems obviously should take into account the required and provided capabilities of each task and resource, respectively.

The most primitive logistics-inspired analysis models in this category are bin-packing and knapsack problems. In the generalized bin packing problem [61, 21], a finite set of items must be packed into a finite set of heterogeneous bins, characterized by possibly different volumes (capacity) and selection fixed costs. In the multi-dimensional knapsack problem [192], the resources are multi-dimensional, and each item requires capacity from several different resources at once. The problem can be extended to the multi-dimensional, multiple-choice knapsack [252], where there are multiple configurations of each task requiring service, and the controller must select a run-time configuration to service the task.

Whereas sequencing decisions may partition tasks into time buckets, resource assignment partitions tasks among the available or required resources, and many resource assignment problems can use bin-packing heuristics to enhance modeling and solution procedures; examples of this are machine scheduling [58] and more generally resource constrained scheduling [103]. To manufacture a product with a given process plan, the assignment of resources to the required process plan steps is considered scheduling [170], and the schedule may give the anticipated start times of each task on each resource [304, 139]. In a confusing over-loading of terminology, routing can be viewed as dynamically choosing at which machine the operation should be performed [298, 44]. Resource-constrained project scheduling generalizes the classic resource assignment problem to support additional resources, where each resource has a limited capacity and each task requires the use of a part of each resource during its execution and the scheduling of project activities is subject to precedence and resource constraints [133]. Powell et al. [210] organize the complexity of resource assignment problems by the increasing degree of coordination required between different types of resources.

3.3.1 Formulation

One common formulation of the resource assignment problem is to aggregate the tasks' capacity requests for a particular process, or capability, and then match the requests to available resource capacity. For example in the transportation problem formulation given in [306], V_{opmt} is the integer number of vehicles of type m with capacity cap_m traversing the arc (o, p) at time t and the total capacity requests, w_l, of the tasks traversing each arc cannot exceed the total capacity of the vehicles on a particular arc, where $X_{lopt} = 1$ if task $l \in \mathcal{L}_t^+$ traverses arc (o, p) in period t:

$$\sum_{m \in \mathcal{M}} cap_m V_{opmt} \geq \sum_{l \in \mathcal{L}_t^+} w_l X_{lopt}, \ \forall (o, p) \in \mathcal{A}, \ t \in T$$

In multi-resource assignment formulations, e.g, such as those described in newsvendor networks (capacitated resource networks) [288] and the resource-constrained project scheduling literature [133], the variable x_j is the amount of process j executed during the period and $A = \{a_{kj}\}$ is the amount of resource k capacity consumed per execution of process j. Therefore the single period constraint is $Ax \leq K$, where K is the vector of available

resource capacities during the period [288].

Aggregate formulations avoid the explicit assignment of a particular resource to a particular task in order to reduce the size of the problem. The formulation given in [108] uses the variable $x_{lt} = 1$ to denote if a resource is assigned to a task $l \in \mathcal{L}_t^+$ at time t, thereby ensuring that each task is serviced by a particular resource. This formulation also ensures that the resource required to serviced the task is in the necessary state and enforcing the constraint that the sum of all the resource assignments to tasks in state, or location, i does not exceed R_{it} the available resources at state/location i in each period t:

$$\sum_{l \in \mathcal{L}_{it}^+} x_{lt} \leq R_{it}, \ \forall i \in \mathcal{J}$$

However, even this formulation does not specify which identifiable resource is assigned to the task, and the controller must then decide which resource in the pool to assign to execute the task. An explicit assignment formulation, such as $X_{mlit} = 1$ if resource $m \in \mathcal{M}$ is assigned to task $l \in \mathcal{L}_t^+$ to execute process $i \in \mathcal{J}$ at time t, allows the solution to the optimization problem to be mapped directly into an executable action.

3.3.2 Applications

While the CVRP is discussed extensively in the context of sequencing visits to customers, little discussion is focused on which transportation resource should be assigned to a particular move task. Golden et al. [109] and Salhi and Rand [234] integrate the fleet sizing and composition problem into the VRP formulation by considering a heterogeneous collection of vehicle options of varying capacities.

In health care systems, the assignment of patient groups to available resources requires knowledge about the capabilities of clinical staff, support staff or medical equipment, and the medical characteristics of patients [178, 208, 142]. In certain situations, patients may need to be assigned to specific nurses (resources) having special skills (capabilities) [213]. Often the resource allocation process is done in a multi-stage process where patients requiring a particular service or resources having a particular capability are scheduled sequentially, thereby ensuring a proper assignment of capability to desired service. For example, Beliën

and Demeulemeester [28] develop operating room schedules in a three stage process: allocation of operating room time to surgical specialties at strategic level, development of a master surgery schedule at tactical level, and then scheduling individual patients at operational level.

Warehouses have many interrelated resource allocation problems where labor and material handling equipment are assigned to load and unload the carrier trucks and pick and put-away products into storage. However, there are also auxiliary resources such as docks, sorter lanes, and storage locations that need to be assigned to trucks, orders, and SKUs to be stored, respectively [229, 115, 226]. Container terminal operations face similar problems where berth allocation is similar to the truck to dock door assignment problem and the quay crane scheduling assigns a vehicle and storage space to unload the container ship [211, 30]. While inventory allocation decisions are often considered in the 'which tasks to serve' classification, the warehouse may have a particular SKU stored in multiple locations, and must decide from which location to retrieve inventory.

Disaster relief operations allocate resources, including distribution vehicles, labor, and relief goods, to transport casualties to medical care [95], distribute goods and resources [123, 200, 20, 222], and other tasks such as structure stabilization, lifeline restoration, and repairing major damage to public works [37, 95]. Yi and Özdamar [306] present an extended CVRP formulation that explicitly decides which transportation mode should be assigned to move goods and wounded people around a disaster area. Resource sharing has proven to be valuable to humanitarian operations [289]: "In Afghanistan, truck capacity was the main bottleneck during the initial stages of the Afghan crisis. With the support of the United Nations Joint Logistics Centre (UNJLC), humanitarian agencies were able to use excess capacity in some convoys, thereby effectively sharing scarce assets (trucks). Shared resources can also be people (skills and knowledge)."

While determining when maintenance should be performed is discussed in section 3.5, once the overhead maintenance tasks are generated they then must be assigned to and executed by utilizing a maintenance resource with a particular set of specialized capabilities

[194, 265]. In maintenance, repair, and overhaul (MRO) logistics, operational decision-making requires trading off the allocation of consumable resource capacity and renewable resource capacity through the joint allocation of available repair capacity among different items and available inventories to field stocking locations [113, 112, 40].

Allocating resource capacity often requires deciding whether to allocate capacity from one resource verses another, in particular sometimes capacity can be 'stored' by producing ahead in anticipation of demand, or in the case of 'make-to-order' manufacturing firms (and service firms) capacity (rather than inventory) is what the firm 'holds' in reserve [18]. Then the real-time decision is to service a task from the stored capacity or active capacity. The multi-resource allocation problem is also evident in newsvendor networks where the decision-maker must decide how to allocate inventory stocks and resource capacity to fulfill market demands [288]. In the extended stock allocation model, the controller is able optionally to delay the allocation of inventory to a customer's order and later use an expedited shipment mode, or drop-shipping mode, to transport parts to the customer [40, 14].

3.4 Problems that deal with where to send a task next

Routing is concerned with determining the route or sequence of operations for each part passing through the system [25, 101]. Often the routing of a part to complete the sequence of required processes is considered planning (a tactical decision) [170, 304, 298]. However in flexible manufacturing applications, selecting a process plan (part routings through the machines) for a particular task supports the assignment of operations and tools to each machine [238, 36]. In machine scheduling, a controller selects a run-time process plan from a predetermined set of available plans [304, 249]. Özgüven et al. [201] addresses flexible job-shop scheduling problems (FJSPs) that encompass routing and sequencing sub-problems, and the FJSPs with process plan flexibility (FJSP-PPFs) that also includes process plan selection as a sub-problem.

The routing problem can also be applied to path selection in vehicle routing problems, where the routing is usually implied to mean sequencing of nodes to visit. In many of these formulations, the "cost of the least cost path from every vertex x_i to every vertex x_j is given

as c_{ij}" [55]. Then at first inspection this formulation seems very different from the process plan selection. However in static and dynamic routing of automated material handling systems, pre-computed routing tables that contain feasible routes for each origin-destination pair are generated a priori and loaded into the controller [168]. State-dependent and event-dependent dynamic routing require the resource controller to gather updates periodically about the system.

In multi-agent systems, the process routes and schedules for a part are constructed or selected through contract net bids [117, 74]. The task allocation and process alternative selection are achieved through the hierarchical bidding processes between involved agents. This method can be applied to determining the optimal route for material flow, where an agent-based framework for an automated material handling system (AMHS) can be organized as a network of 'node agents' connected by unidirectional links, and either the transportation resource or the task queries and negotiates with the node agents along prospective paths to construct the best path based on the current status of the system [168]. Navigating this network of autonomous node agents can be formulated as a sequence of move sub-tasks in the process plan language, and then the controller must select the best process plan to complete the parent task.

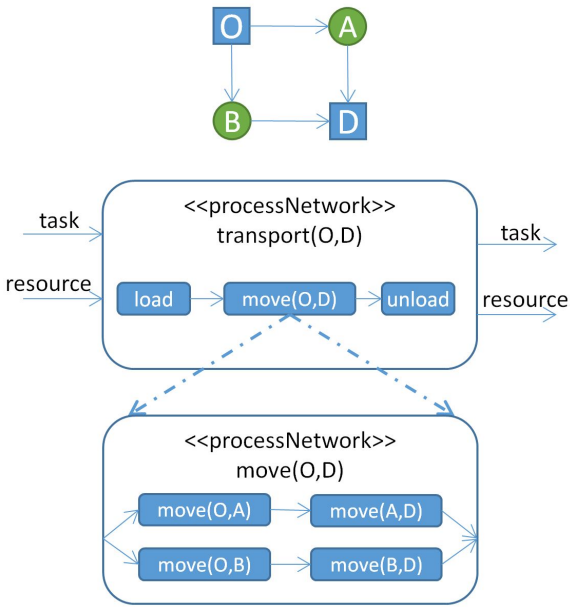


Figure 28: A move task can be decomposed into a digraph of move sub-tasks.

In this formulation the transportation resource is repeatedly asking "where, or which control point node, should I move the task to next?", and a sequence of answers produces a transportation route. In this case, navigating the network is a relatively unconstrained process plan where the structure of the candidate process plans mirror the network structure of the AMHS (Figure 28). If this is a specific case, then the question once again reduces to process plan selection (from a pre-formulated set). This querying process suggests that the routing question is complementary to the admission control, in that it initiates the call for proposal (cfp) that is the input argument of the admission process (cf. section 3.1).

3.4.1 Formulation

The flexible job shop problem (FJSP) provides a complete mathematical formulation of the routing and process plan selection problem [201]. In the FMS extension, Gamila and Motavalli [101] consider the additional sub-problem of assigning tools to machines.

The FJSP consists of a set of n independent tasks $\mathcal{L}_t^+ = \{\ell_i\}_{i=1}^n$, each having its own processing order through a set of m machines $\mathcal{M} = \{m_k\}_{k=1}^m$. A number $p(\ell_i)$ of ordered processes $(O_{i1}, \ldots, O_{ip(\ell_i)})$ has to be performed to complete task ℓ_i. Process j of task ℓ_i, (O_{ij}), rather than having to be processed on a predefined machine $m_j \in \mathcal{M}$ as in JSP, can be processed by any machine in a given set $M_j \subseteq \mathcal{M}$. This formulation should also be extended to support multiple resource requirements.

The FJSP-PPF considers multiple process plans for tasks by excluding the final assumption of FJSP [201], i.e., there is only one feasible process plan for each task. The problem consists of a set of n tasks, each having a set of $\tau_{(\ell_i)}$ process plans $\mathcal{P}_{\ell_i} = \{\rho_{ip_{(\ell_i)}}\}_{p_{(\ell_i)}=1}^{\tau_{(\ell_i)}}$. The process plan $\rho_{ip_{(\ell_i)}}$ of task ℓ_i is an ordered list of $p_{(\ell_i)}$ processes. It is assumed that the process plans are known in advance and represented by linear precedence relationships. Process p_j in the process plan $\rho_{ip_{(\ell_i)}}$ of task ℓ_i, $(O_{ip_{(i)}j})$ can be processed by any machine in a given set $M_j \subseteq \mathcal{M}$. Because only one of the alternative plans is to be adopted for each task, the FJSP-PPF deals with not only routing and sequencing sub-problems but also the process plan selection sub-problem: choosing a process plan for each task ℓ_i from a given set of $\mathcal{P}_{\ell_i}$, assigning each process $O_{ip_{(i)}j}$ to a machine selected from the set M_j and ordering

tasks on the machines so that $c_{\max}$ is minimized.

3.4.2 Applications

In centralized control structures and local routing decisions, the controller can use nearly complete information to solve a modified version of the resource assignment problem to select a process plan. However, more general routing problems arise in decentralized and agent-based or holonic control architectures. In these decentralized schemes, a local controller or a task holon that is negotiating a path through its process plan, does not have global influence or knowledge. This is accomplished by tackling the "dual" to the problem that a resource must decide which tasks and how to service them (figure 29). In this example, there are three tasks in the system, which each require two processing steps. Process 1 can be executed on either machine *m1a* or *m1b*. Then each task maintains a process plan (with machine flexibility) which is complementary to the machines' process plans of tasks to service. The complexity of maintaining these separate, but complementary views of the system and solving the scheduling problem highlights the challenges that arise in multi-agent systems.

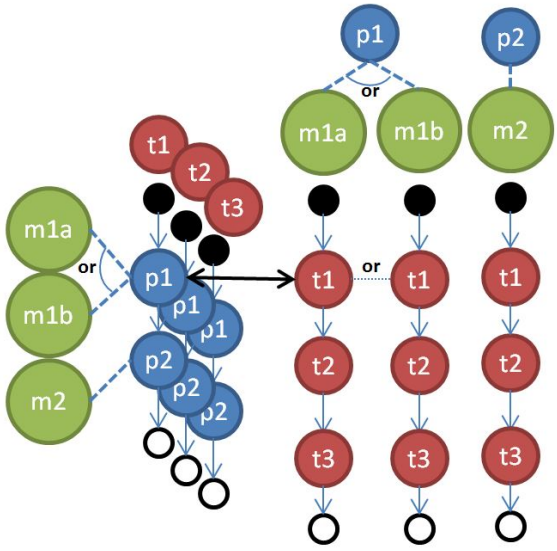


Figure 29: Each task and resource maintains an independent process plan of the sequence of processes it must execute. The process plans of a particular resource and task intersect when the task is processed on the resource.

For patients within a health care system, treatment typically consists of multiple stages

and the composition and sequence of these stages defines the route of a patient. Process plan patterns or templates are constructed by identifying different patient types and designing customized patient routes for each type, thereby preventing superfluous stages and delays [142]. More generally in a multi-skill, multi-class system containing multiple pools of servers and customer classes having different characteristics, skills-based routing is the real-time routing of customers to available servers who have the capability to serve that customer class during the operation of the system [180].

In warehouse order picking, batching, routing, and sorting strategies may offer several different ways, or process plans, to construct a particular shipment [207, 115, 67]. There may be several locations containing a particular SKU and multiple pickers available to pick the line item from stock. In batch-pick and sort systems, an additional sort step is required. There may be additional extensions to construction out-bound warehouse shipments, but these formulations result in a multiple, complex process-plans for assembling each order.

Many disaster relief and humanitarian logistics problems formulate the distribution of goods and services as a CVRP [200, 294], and Barbarosoğlu et al. [22] consider the routing of helicopters from the operation base to disaster points in the emergency area as part of their decision-support system.

3.5 Problems that deal with when to change the resource state

The rate of service can be controlled by setting the number of available servers, or the state of each of the servers, and the optimal operating policy consists of rules for turning the server on and off that result in the lowest long-run cost [156, 270]. In the context of capacitated resources, this can be interpreted as adding capacity to the system, but for discrete-state resources it can be interpreted as changing the state of the resource, or capability, from idle to not idle.

In a unifying formulation of this class of control questions, the controller must decide when to take an action to change the state of the resource but also must decide which state to change it to, e.g. how much additional capacity or to which set-up position. Furthermore once an action has been decided upon, the controller must generate an overhead task for

the resource state change to be executed. These overhead tasks may be inventory orders, maintenance tasks, set-up tasks, etc.

3.5.1 Formulation

There is a limited amount of literature that explicitly models the state and transitions of the system. Rather than explicitly specify the state of the resource that will service a particular set of tasks, one is forced to infer the changes to the state of the system, i.e. if task i is of type 1 and task j is of type 2, and i follows j, then it is assumed that the machine changed setups in between processing epochs and incurred a time-penalty of s_{ij}.

In switching curve policies, in order for the system to move from one state to the next an overhead set-up task must be generated and scheduled in order to produce the next product. In simulation, the controller examines each arriving entity and generates an overhead set-up task if the machine cannot accommodate the required processing capability of the arriving task. Multi-class switching curves explicitly denote which state the system should be in [121, 68]. In this formulation, $X(t)$ is the inventory or back-order state of the system at time t:

$$C_a(t) = C_a(X(t)) = \begin{cases} 0 & \text{when the action is to idle,} \\ 1 & \text{when the action is to produce type 1,} \\ 2 & \text{when the action is to produce type 2.} \end{cases}$$

In this case, it's assumed that there is only a single resource or all of the capacity, or resources, are changed to support this production plan.

However, this assumption can be relaxed to allow for partial capacity changes or changing multiple resources at once: $y_{ijt} \in \mathbb{Z}$ is the quantity of resources, such as vehicles, inventory, and empty containers respectively, re-positioned from location i to j beginning at time t [108, 6, 60]. This complements the assignment constraint given in section 3.3

$$\sum_{j \in \mathcal{J}} y_{ijt} - \sum_{l \in \mathcal{L}_{it}^{+}} x_{lt} = R_{it}, \ \forall i \in \mathcal{J}$$

While this formulation explicitly captures the re-positioning of resources with a dedicated variable, it treats the capacity change at an aggregate level. Nevertheless, i and j can be generalized to any state space, e.g. locations, inventory levels, process set-ups, etc. Once the

decision is made to re-position inventory or empty resources, an overhead task is generated for the system which then decides when and how to execute the task.

As with the resource assignment problems discussed in section 3.3, while aggregate formulations are convenient to limit the problem size, modeling the resources individually allows a more natural mapping from the solution of the problem to the action that the controller executes. Then the questions then are when, which resource, and to which capability should it be changed: $X_{ijt}^{k} = 1$ if resource k is changed from state i to state j at time t.

3.5.2 Changing Resource Capability

Each resource in the system may provide multiple services to the system, e.g. a machine is capable of executing multiple processes and producing several different parts. However, these resources incur a cost to change their state in the form of a set-up time or a re-positioning cost. The intention of this section is to draw parallels between setups (on a machine) and re-positioning a vehicle, which includes moving from its current location to the origin of its next task. These 'change state' actions can be either anticipatory or non-anticipatory or in some cases a partial move (to a central location or neutral state) can be made to reduce response time in the future.

In the make-to-stock production control problem [68, 149], the controller must decide dynamically when and which part type to produce. The challenge of this problem is the cost associated with changing the set-up (state) of a particular machine to accommodate a particular part type. In dynamic fleet management [108], the state of each vehicle is specified as a geographic location, and a task can only be assigned to a vehicle in a specified subset of states; i.e. a vehicle can only service a customer's service request if it is in same location and a resource repositions, i.e., perform the 'set-up process', by moving empty units from one location to another. When the state of a 'mobile' renewable resource includes a geographic location, or address, then the resource can be constrained to return to a 'starting' location [77]. More generally any resource can be constrained to returning to a particular state before beginning a new set of tasks; e.g. machines to a particular setup, robotic arms to a default position or configuration, AS/R vehicles returning to an optimal dwell point,

etc.

One way to improve quality of service is to balance the allocation of tasks to resources, but in some cases there is the option to re-position capacity from one location (resource) to another; e.g. Paterson et al. [205] provides a review and taxonomy of transshipment policies, including classifying policies as proactive or reactive. White [300] and Crainic et al. [60] discuss transshipment of empty shipping containers to support tasks that can't be serviced at their origin due to the lack of a shipping container (resource). Dejax and Crainic [70] provides an overview of problems related to re-positioning empty containers and vehicles in a logistics network.

3.5.2.1 Anticipatory Moves and Pre-positioning

Potts and Kovalyov [209] discuss anticipatory and non-anticipatory setups, where a (non-anticipatory) "setup preceding the processing of some batch cannot start on the current machine before all jobs of this batch are released and have completed their processing on a previous machine". More generally, an idle resource may be allowed to make an anticipatory move to better position itself for the next service request, which exhibits both spatial and temporal uncertainty [176]. Several authors discuss optimal dwell point policy for automated storage/retrieval systems in warehouses [229, 203, 226]. A priori planning and pre-positioning capacities of key resources is critical in managing response efforts after an event; see, e.g. ambulances in emergency care services [142], humanitarian logistics [221, 289], and military logistics [221].

3.5.2.2 Tooling Selection in FMS

The tools contained within a flexible machine's tool magazine determine which particular capabilities the machine is able to offer, and in most systems, all the required tools must be acquired prior to processing a part [134]. Sharing of tools by machines effectively increases the capacity of tool magazines and eliminates the movement of parts from machine to machine to search for a particular tool or capability [162].

The tool selection problem also incorporates the re-positioning decision, too, since the

tools themselves are shared and re-positioned between the machine tool magazines, intermediate tool storage, and central tool storage [89]. Re-positioning tools and the cycling of spare tools influence the system reliability and the potential for parallel processing of identical parts on several machines; however increasing the number of identical tools loaded simultaneously into the tool magazine of a single machine reduces its product scope, but also reduces the number of setups needed to change worn tools [112].

Much like setups and re-positioning action, the time required for a tool switch may be significant relative to the processing time [134], therefore there is a trade-off in reducing the down-time for particular set of capabilities (increasing the effective capacity) verses increasing the capability set. Furthermore, once the decision has been made to re-position certain tools there is an additional decision of which transportation resource will execute the task [272].

3.5.3 Changing Operational Capacity

Whereas the previous subset of 'change-state' questions were related to changing the capability of a particular resource, this question also relates to changing the capacity of a particular resource, including maintenance, working overtime or outsourcing, and inventory replenishment and reordering. From the perspective of the controller, the questions are 'When to adjust the operational capacity?' and 'By how much?'. While the behavioral and execution mechanisms may vary in each of these questions (unlike the other questions where the mechanism is the same), the result of each of these decisions is that the capacity of the targeted resources is increased (or decreased) over some period of time. In [90], renewable resources are a class of resources that have their capacity increased by certain designated processes; cf. the *increaseCapacity()* and *decreaseCapacity()* methods of the capacitated resource class in section 2.3, figure 18.

3.5.3.1 Capacity Renewal & Inventory Replenishment

The inventory replenishment process requires jointly deciding when and how much additional inventory is required [13, 290]. This is actually a general statement for formulating a

consumable resource capacity replenishment problem: one variable denotes a reordering action (when) and a second denotes how much additional capacity is obtained by that action. Angelelli and Speranza [10] propose a periodic vehicle routing problem with intermediate facilities, where vehicles can renew their capacity. Due to the lack of established infrastructure, disaster relief logistics face a modified problem for re-fueling each helicopter (or any transportation resource) at a centralized operation base [22]. The restocking decisions for the forward pick area in a warehouse address how frequently and at what time should the inventory for an SKU be replenished and how should each SKU be distributed and moved (redistributed or replenished) among the different storage areas [229, 115, 226].

3.5.3.2 Maintenance

Maintenance is the control process of deciding when to sacrifice capacity in the short-term to gain capacity in the long term. Preventative maintenance can be run-based or time-based and corrective maintenance is failure-based. Each of these processes function to restore or improve the operating state of the resource, and often the maintenance action is considered to trigger a renewal period for the machine's capacity, i.e. it is returned to a 'good-as-new' state [43]. Once again the decisions are when the maintenance action should occur and how much should be done, which implies how much additional capacity is obtained from the action. The availability of the machine can be optimized by selecting appropriate preventive maintenance rates $\omega(t)$ or maintenance intervals τ which decreases the corresponding machine failure frequency and/or duration [105, 43].

The aircraft maintenance-scheduling problem not only determines when and where maintenance will be performed on an aircraft, but also, if necessary, which other aircraft (resource) will be assigned to complete its assigned tasks while undergoing maintenance [265, 247]. Like other overhead tasks generated by the system, the maintenance task is then scheduled according to maintenance policies and executed by utilizing (dedicated) maintenance resources [194, 43].

3.5.3.3 Overtime and Outsourcing

To deal with an over-loaded production system, a controller may seek additional production capacity by authorizing overtime, hiring temporary labor, or by outsourcing some tasks [86, 132, 291]. While the quantity of regular capacity units is the result of a long-term strategic decision, this additional "non-regular" capacity can supplement the system's regular capacity in the short-term, though typically incurring significantly higher costs. For example, a best practice in health care systems is to adjust staffing levels on the day of surgery by asking nurses to work overtime or getting help from qualified nurses of other departments, since the benefits of having scheduled cases performed outweigh the costs of working overtime [79].

In agent-based systems, the outsourcing process generates a call for proposal to solicit new resources to join the system to supplement existing capacity, provide new capabilities, or handle one particular task. In addition to deciding when to authorize outsourcing, the controller must decide how much work to allocate to subcontractors and which jobs to subcontract [302, 29].

3.6 Separability and Joint Problem Classes

Up to this point, what has been described in the previous sections are considered 'atomic' control questions, which can be combined and solved jointly in some cases. In many solution approaches, the scheduling problem is decomposed into the resource assignment problem and the sequencing of tasks on a resource, and then solved hierarchically [25, 35]. Alternatively, the decomposition can focus on selecting a process plan for each part and assigning manufacturing resources for specific time periods to the set of manufacturing processes in the plan [36, 249]. Dauzère-Pérès and Paulli [63] use a digraph approach to integrate the reassignment and resequencing methods by moving connective arcs around in the graph to jointly accomplish both control decisions simultaneously. However the authors state that 'in all the approaches we are aware of, the assignment of operations to machines and the sequencing of operations on the machines are separated. This separation is done either directly by considering independently assignment and sequencing, or in a local search

algorithm in which reassignment and resequencing of an operation are considered as two different types of transitions." [63].

In the inventory routing problem [92], the system has two resources, inventory stocks and delivery vehicles, and three decisions have to be made: (1) When to serve a customer, (2) how much to deliver to a customer when it is served, and (3) which delivery routes to use. Their decomposition can be interpreted as separating the sequencing, assignment, and routing decisions. After the controller decides to service a particular customer, it must assign a vehicle to do so [108]. In [131], the variable neighborhood search procedure for the periodic VRP hierarchically organizes the (periodic) pattern selection, route assignment, and route sequencing procedures.

The order acceptance problem can be extended to incorporate scheduling decisions at the same time [66, 259, 227, 258]. Akkan [7] suggest a policy that accepts orders only if they can be included in the schedule, such that it is completed before its due date, and without changing the schedule for already accepted orders. In health care logistics, the scheduling decision is decomposed into three decisions: patient selection, assignment of patients to the staff, sequencing of patients within a time period [197].

The following scheduling formulations capture the resource assignment and task sequencing decisions [297, 33, 181]:

1. $y_{irk} = 1$, if job i is scheduling in the r^{th} position for processing on machine k
2. $y_{ikt} = 1$, if job i is processed by machine k during period t
3. $y_{ii'k} = 1$, if job i precedes job i' (not necessarily immediately) on machine k

3.7 Summary

While the domain-specific language (DSL) discussed in the previous chapter is sufficient for facilitating interoperability between system models and a broad range of analysis models, the control questions discussed in this chapter refine the usage of the DSL by mapping a specific subset of analysis models, specifically optimal-control models, directly to the functional specification of the operational control mechanisms. That is, each control question maps to

a decision variable in the corresponding optimal-control analysis models that are constructed to answer the question.

Why does the operational control literature solve these particular questions? A reasonable conjecture is that collectively, the control questions discussed in this chapter define a comprehensive functional specification of all the decision-making mechanisms that a DELS controller needs to be able to provide; i.e the set of interventions that a controller may take upon the system to influence the performance of the system.

This chapter organizes the many representations and descriptions of operational control found in literature into equivalence classes of control questions that share a common functional definition and formulation which suggests that these questions form a canonical set of control questions for operational control in DELS.

This fundamental functional specification of DELS control behavior is formalized using the abstract PPRF semantics discussed in the previous chapter. By applying the same encapsulation process as described in section 2.5.1, a standard representation of each class of problems enables not only a uniform interface to solution tools, thereby creating opportunities for interoperable, or plug-and-play, analysis tools; but also the specification of a uniform behavioral mechanism that the system uses to execute the results of the optimal-control analysis model.

CHAPTER IV

A METAMODEL OF OPERATIONAL CONTROL

"Management is *event-driven* and performed with *discrete* activities, where the continuous service-provision with a certain quality, as it might be expressed on the requirements level, must be translated into discrete management activities performed by managing objects on managed objects" [157].

Each logical DELS node has a controller, and there are several possible architectures for the controller itself [65, 268, 269, 182, 232]. In [65], the controller has four components: assessment, optimization, execution, and monitoring. In [268], the knowledge of the scheduler in a case-based reasoning system is abstracted into four components: 1) state recognition, 2) policy selection, 3) policy implementation, and 4) policy execution. In [269], the supervisory reactive scheduler performs the following functions: on-line monitoring and analysis of shop floor status; modification, adaptation, and repair of existing schedules; decision making, i.e., predictive/reactive scheduling; execution of scheduled activities; and learning from experience during problem solving. In agent-based fractal architectures [232], the basic fractal unit consists of five functional components: an observing module, an analyzing module, an executing module, an organizing module, and a reporting module. In agent-based holonic control architectures, each controller is divided into a high-level control and a low-level control [182, 296]. The higher-level control is capable of intelligent decision-making, communication, and cooperation. The lower-level control is run on a programmable logic controller (PLC) or embedded controller and is focused on the real-time execution of tasks.

These controller architectures are constructed primarily to address scheduling in manufacturing with limited extensibility to support more general system descriptions or analysis methods, but they suggest a common structure that should be formalized into a DELS controller reference architecture. The functional architecture of the controller can be derived through a question-driven process, where the goal is to identify "what the controller

must do" rather than worry about its particular implementation. The *Monitoring Function* handles "When does a question about the system need to be answered and which one?". Then there is a corresponding *Decision Support* function that must decide "How to answer the question?". This question and its corresponding function can be further decomposed into "What problem needs to be solved to answer this question about the system?" (formulation), "What is the solution to the posed problem?" (optimization), "How do you implement the solution?" (implementation), and "What action should be taken as a result of this answer?" (execution).

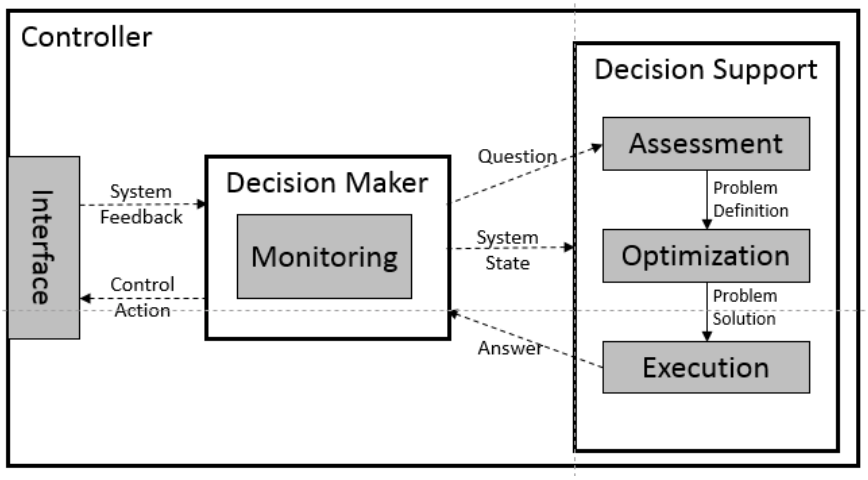


Figure 30: The Controller Functional Architecture.

The canonical set of control questions identified in chapter 3 define a comprehensive functional specification of all the control mechanisms that need to be supported by the controller. Development of a controller that has automated access to the wide variety of analysis models and solution methods needed to support operational control decision-making relies on two capabilities. First, there must be interoperability between system models and analysis models. Second, there must be a consistent interface to formulating and solving analysis models and then extracting the results into executable actions (or behaviors). This second component is critical to bridging from OR analysis models to practical applications of the methods and tools, and it is often neglected in the operations management research literature.

To address this gap, this chapter constructs a round-trip analysis methodology that

provides a more fundamental insight into how analysis is conducted: formulate the analysis model, solve it, then implement the results as plans and policies, and finally execute the prescribed control action. Answering the control question first requires formulating the optimal-control analysis model to be passed across an interface to the corresponding analysis tools contained in the optimization module (Section 4.3). The process of formulating the optimization analyses strips the domain semantics from the instance data before solving. The resulting solution is semantic-free as well and must be translated back into domain-specific semantics. This translation process coverts the optimization results into executable plans and policies (Section 4.4). Finally, these plans and policies are returned to the decision maker and configured into an execution mechanism, such as a finite-state machine or production rule system (Section 4.5). Since the focus of this dissertation is turning feedback into actions, which forms the functional description of the decision support module, this chapter will begin with a brief overview of the interface definition, including a formal definition of tasks and services, and the monitoring function that receives and organizes the feedback from the system.

4.1 Interface Definition

The context of a system defines the interface and interaction of that system with its environment, including receiving feedback from its owned resources and subordinate controllers, relaying control actions that should be executed by subordinate controllers, and communicating with other controllers in the system. Message-based communication protocols, such as the contract net protocol, standardize the syntax of the messages and provide an interaction protocol between agents [261]. Additionally, the abstract architecture requires a transport-message-service that is responsible for sending and receiving messages between agents.

While these messaging protocols standardize the structure and syntax of the message, the content of the message is specified by an ontology, encoding, and a language along with the content itself. For example, the Computer Aided Manufacturing using XML (CAMX)

standard defines an event-based conversational framework based on the exchange of standardized XML messages related to process events and to control commands [84, 73]. In particular, the IPC-254x series of standards define unidirectional messages associated with shop-floor equipment events. The IPC-255x series in turn defines messages for bidirectional communication of supervisory control functions between equipment and upper level controllers. The CAMX and CMSD standards also define collections of events to be expected from the manufacturing floor, including equipment state changes, item events, equipment flow events, and equipment events.

While CAMX specifies a message-passing framework for manufacturing environment, there remains a need to define a content specification language that is applicable to the broader DELS domain. This content specification language should include the events that are broadcast, the services that are offered, the information contained within the tasks, such as product and process definitions, and the actions resulting from the controller's decision making process. The rest of this section will construct semantics for task as the unit of work (section 4.1.1), services to encapsulate advertised process capabilities (section 4.1.2), and an event definition language as part of the broader monitoring function (section 4.1.3).

4.1.1 Tasks

The *Task* is the logical information class of authorization that contains orders, jobs, etc. The Task itself is a subtype of *TokenAggregation*, which itself is a type of Token that allows other tokens, such as order documents, to be treated as a single unit [273]. The Task's Bill of Tokens can be partitioned into tokens that are required by the PLANT, such as resource tokens and material tokens, and tokens that are required by the CONTROL, such as information tokens, order document tokens, and plan tokens. Task extends the notion of the basic token by specifying a *requiredService*, the Process that needs to be executed to satisfy the *Task*. The referenced Process gives a natural decomposition of the Task into *sub-Tasks*. This decomposition associates a subtask with each of the processes contained in the *nestedProcessNetwork*. Explicitly, these subtasks will inherit the *SequencingDependencies* from the *nestedProcessNetwork*.

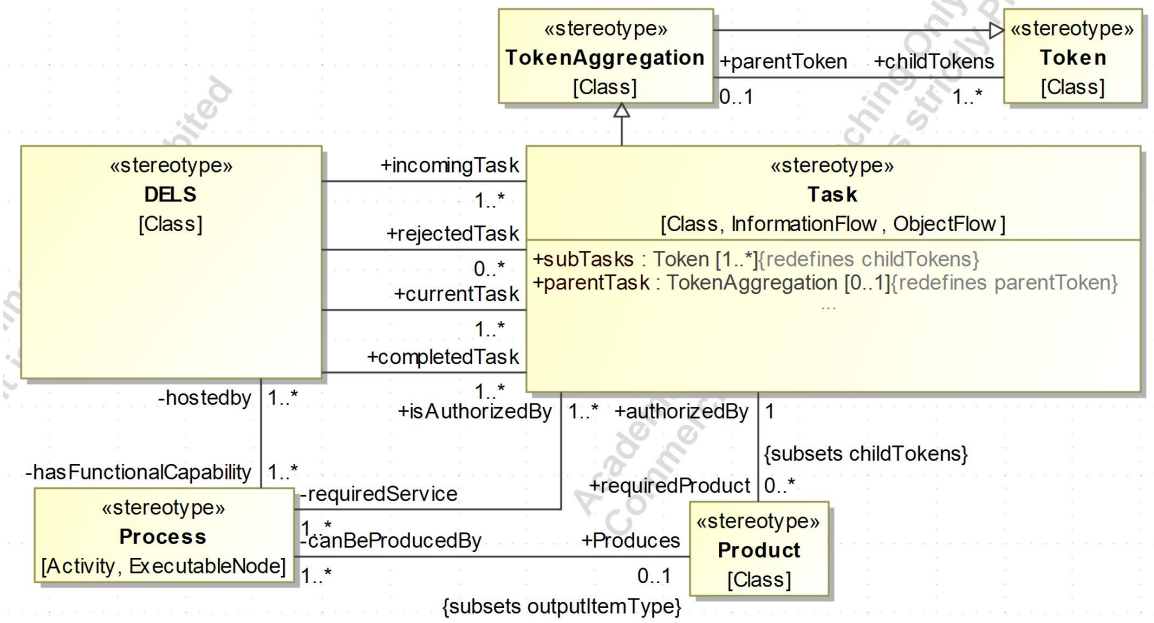


Figure 31: The Context of the DELS is given by the flow of Tasks in and out of the System.

The semantics of Task, defined both abstractly and recursively, are intended to clarify the semantic ambiguity associated with statements such as 'the work order authorizes a job which consists of a series of operations', because it is unclear whether jobs can be aggregated into jobs or operations can be decomposed into jobs as well as the implicit polymorphism between logical authorizations and physical order documents. In the DELS semantics, a task authorizes a process to be executed, which can be decomposed into subtasks authorizing nested processes, recursively ad infinitum.

Consistent semantics that allow tasks to be aggregated and decomposed are important for self-similar and uniform controller architectures where the resource set can be ever changing, resource clusters can be formed to address a particular task, or in agent-based systems where "[the] agents can subcontract tasks to other agents, a process that involves breaking a task in a number of sub-tasks handled by different agents, or clustering a number of tasks into a super-task" [236].

In addition to authorizing the Process specified by the required Service, any resulting Product becomes a member of the corresponding Task's Bill of Tokens upon being produced; i.e. it is a subset of the TokenAggregation's childTokens. This suggests that

multiple token types may be eventually added to the Task's final Bill of Tokens, including the physical Product and any supporting documentation, including inspection documents, shipping documents, etc.

A system's currentTaskSet at any particular time t is defined as $\mathcal{L}_t^+ = \{\ell_1, \ell_2, \ldots, \ell_n\}$ (cf. section 3.1). $\hat{\mathcal{L}}_t$ is the set of *incomingTasks* that first become available in time period t and $\mathcal{L}_t$ is the set of tasks available at time t before new arrivals are added to the system. Then $\mathcal{L}_t^+ = \hat{\mathcal{L}}_t \bigcup \mathcal{L}_t$ is the set of *currentTasks* available to service at time t [108]. Suppose the tasks needed to be explicitly accepted first before being added to the system, then this formulation is extended to selecting $\mathcal{L}_t^* \subseteq \hat{\mathcal{L}}_t$, and $\mathcal{L}_t^+ = \mathcal{L}_t^* \bigcup \mathcal{L}_t$ is the set of tasks available to service at time t. This also implies that $\hat{\mathcal{L}}_t \setminus \mathcal{L}_t^*$ is the set of *rejectedTasks* in period t.

4.1.2 Service

In this section, the basic network definition of the TFN is extended to support active relationships between DELS nodes, which defines how the system interacts with the other systems in its ecosystem (figure 32). The relationship, or association, between two DELS nodes is specified by a *Contract*. The result is a syntactical interface to research on contracting, which connects implementation-focused models of relationships, such as service-oriented architecture (SOA), to soft models of relationships, such as the contracting literature from the operations management domain.

In a SOA, service providers are defined by the functions, or services, that they provide. In DELS, the capability to perform a service, or execute a function, is defined by the resources, and their capabilities, available to the controller (Section 2.2 & Figure 12). Service-oriented architectures "provide a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations" [1]. In ordering a service (a pre-defined behavior), there is an expectation of certain attributes of the service (cost, time, performance, etc.), and in general, the customer or client is not concerned with how the service is executed, but merely waits for a response regarding fulfillment or rejection of request.

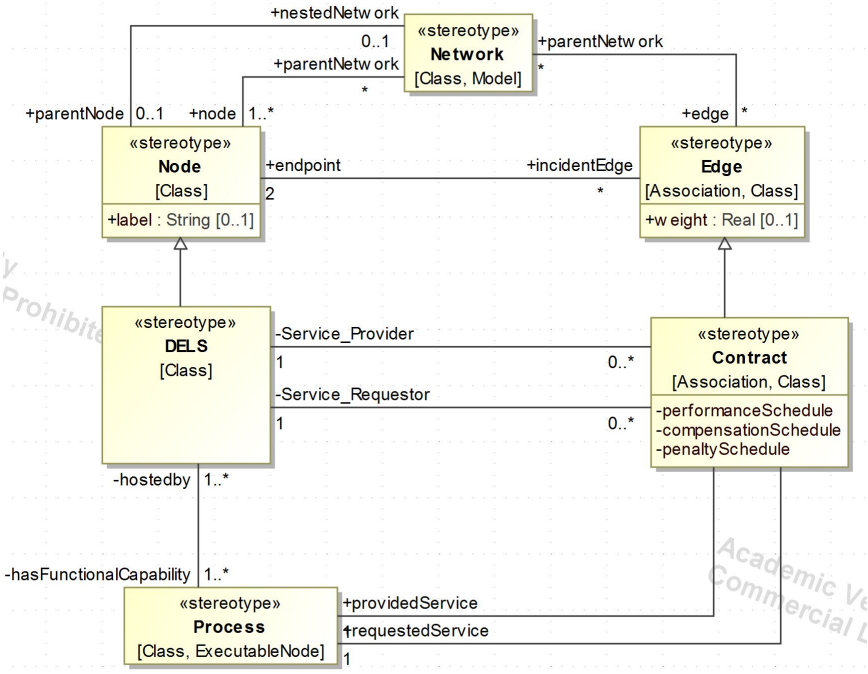


Figure 32: SysML model of DELS supporting contract for a service.

The concepts of contracts and negotiation arise in the supply chain management literature [279, 233, 148], where the literature typically focuses on the negotiation process and achieving system coordination. Typically, game theory is used as the mechanism to resolve negotiations and design contracts without specifying how that mechanism is implemented [91]. And in fact, extensions to the contract net protocol can specify the bidding and negotiation protocol, including dutch auction mechanisms, among others [26].

The semantics derived from service-oriented architectures provide an interface to more mature research governing the relationships between agents in a distributed system. In the distributed decision-making literature, most interaction mechanisms are modeled using some variant of the contract net protocol (figure 33). The contract net protocol [261] is a high-level communication protocol for facilitating the distributed control of cooperative task execution. Within the contract net protocol, each message requires specifying a content-language that is used to express the content of the communication between agents. The DELS DSL (section 2) provides semantics for the task description (what needs to be done and who is eligible to do it) and the contract net protocol provides a useful set of semantics for specifying how the tasks are distributed among the processing nodes through a bidding

and negotiation process.

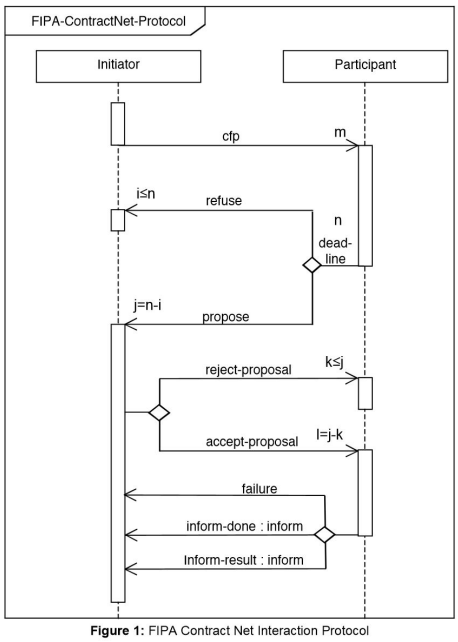


Figure 33: Contract Net Interaction Protocol

However, each processing node encounters the same high-level control problem, i.e, which tasks to bid on, and how to optimally execute the tasks it has won. Therefore, additional semantics are required to describe the architecture of the negotiating parties (DELS controllers), the coordination and negotiation protocols, and the communication protocols. Faratin et al. [91] extends the service-oriented negotiation framework and specifies what negotiation protocol will be used, what issues require negotiation, and what reasoning model the agents will employ. As discussed in section 3.1, this dissertation extends the contract net protocol literature to provide an interface to decision making mechanisms that are capable of more general bidding and negotiating strategies; e.g. which tasks to bid on and how much to bid? Where to schedule the tasks and which resources to subcontract to complete the required service capability?

When more general strategies are incorporated into the controller architecture, several open issues arise, such as identifying the analyses that would be required if the negotiations were utility maximizing agents that may seek a schedule of prices and lead-times. While in most cases the contract net protocol's formalization of the call for proposal (cfp) method is

restricted to the common case of proposals characterized by a single parameter (x) in the proposal expression, this can be extended to support multiple proposal parameters, demand curves,etc. Typically these negotiations only focus on a single request for service, which results in a single task while ignoring the potential for long-term strategic contracts. Also, most negotiations only focus on compensation and lead time, ignoring the potential for the service level and penalty schedule to be the targets of negotiation.

4.1.3 Events and the Monitoring Function

In order to make a clear statement about the nature of the problem, the monitoring function must define the types of disruptions, describe the context, and define the actions that can be taken in the face of the disruptions [15]. The monitoring function is defined by the functional responsibility to listen and respond to feedback from the DELS' environment, i.e. 'What events am I listening for?', and 'What do I do in response to a particular event?'.

From a formal perspective, what is produced, published, propagated, detected, and consumed is a (typically asynchronous) message called the event notification, not the event itself, which is the state change that triggered the message. The monitoring function establishes a listener, event handler, and a message queue to receive feedback from the external world in the form of clearly defined classes of events and targeted messages. From the distributed database management literature, the Event Definition Language [157] provides semantics to specify events that can arise from monitoring the system (figure 34); three sub-types of monitoring events are defined: polling, notification, and timing events. The Reaction RuleML specifies three rule styles for reaction rules: active, message, and logical reasoning [204].

Run-time verification, or run-time monitoring, consists of a software module, an observer, that monitors the execution of a program to ensure that it is running correctly with respect to a formal requirements specification [171, 24, 59]. The Monitoring and Checking (MaC) architecture [171] uses two different event specification languages: the Primitive Event Definition Language (PEDL) and the Meta Event Definition Language (MEDL) (figure 35). The PEDL is a language for writing monitoring scripts and defines the monitored

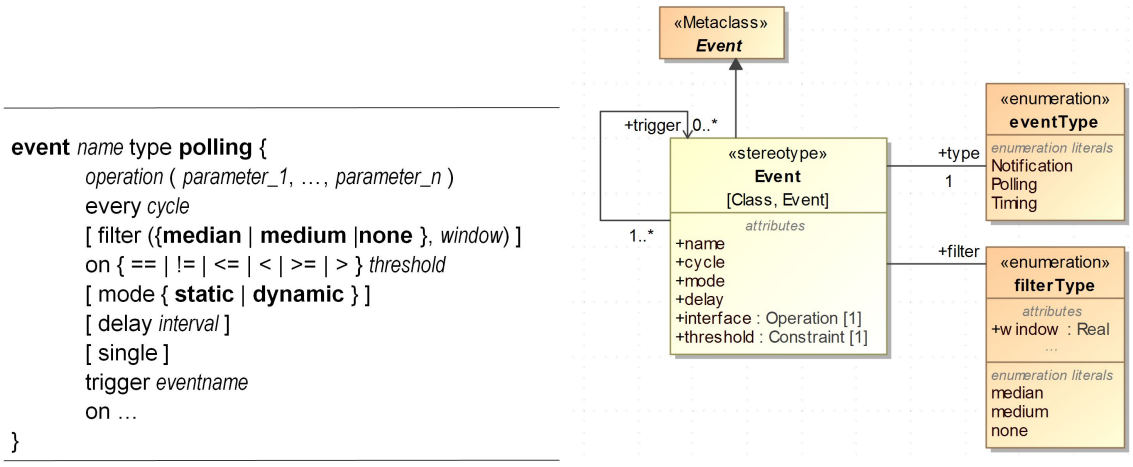


Figure 34: Event Definition Language Structure [157]

entities, the defining conditions (over the monitored variables), and the defining events. The MEDL uses these definitions to specify the requirements for and constraints on the monitoring function in order to evaluate events coming from the system. Colombo et al. [59] provides an overview and comparison of a few run-time verification tools.

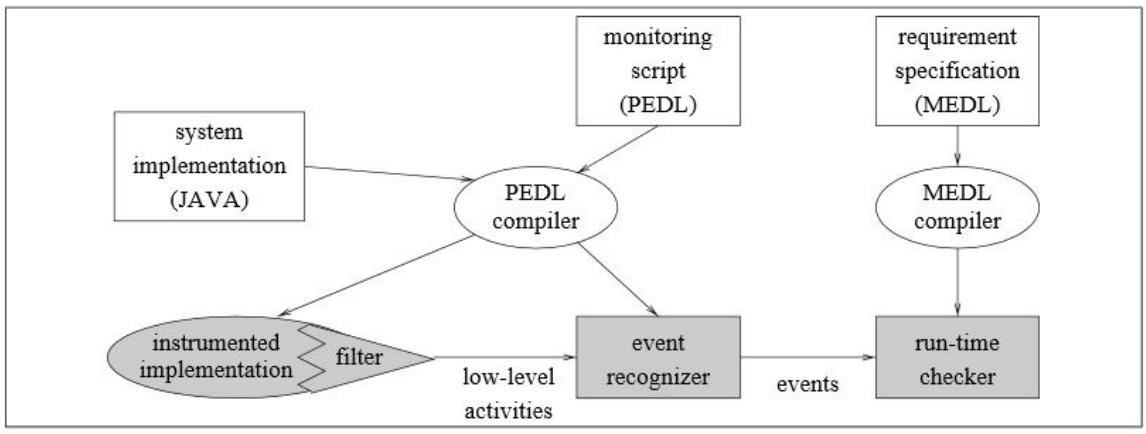


Figure 35: Overview of the MaC framework [171]

Since the event recognizer and run-time checker are automatically generated (figure 35), although before run-time, each resource and task would define and provide the system with a set of pre-defined events; e.g. break-down, completion, delay, etc., which the monitoring function then integrates into the set of events it's listening for and updates its event recognizer, run-time checker, subscriptions, etc. Therefore, resources and tasks must be pre-configured with a library of events that conform to a standard event definition

language.

Then, deciding what actions should be taken as a result of the observed events and system feedback is governed by rules and executed by finite state machine and production rule systems (section 4.4), which will be discussed more in depth in section 4.4.2.1 after rules and policy-based management systems are introduced.

4.2 Decision Making: Performance, Motivation, and Authorization

"For any [control] activity the [controller] must be **motivated** and **authorized**. Both aspects are generally independent and must be described separately." [157]

What motivates the controller? In this section, requirements, goals, and utility or value are presented for specifying the motivation of the DELS controller. These three types of motivation of a DELS are ordered in decreasing degree of restrictiveness and increasing degree of autonomy for the controller to make decisions. The objective of this section is to derive a standard representation of a DELS' motivation which can be translated into an 'objective function' for any corresponding optimal-control analysis model. This section will start with a short overview of these decision making mechanisms, then section 4.2.1 formalizes the language elements for specifying DELS performance attributes and their associated motivation, and then the authority continuum will be discussed in three sections: requirements (section 4.2.2), goals (section 4.2.3), and utility (section 4.2.4).

Suppose the Controller is instantiated for a manufacturing job shop, and the two performance attributes that the controller must balance are total labor cost and on-time delivery. In this first system these performance attributes are specified as hard, inviolable requirements, i.e. if the task is not completed on-time, the customer reneges on the request or abandons the system. Then the motivation may be specified as the following *constraints* on weekly labor cost and on-time delivery, which the DELS is *not authorized* to violate.

1. The system's total labor cost (weekly) shall not exceed $ 8,000

$$aggregation(DELS.LaborCost, `weekly') \leq 8000$$

2. Every task shall be completed by its due date

$$finish(t) \leq \ell.duedate \; \forall \ell \in \text{DELS.taskSet} \iff DELS.onTimeDelivery = 100\%$$

Despite sophisticated planning and scheduling analysis tools, sometimes in stochastic systems these constraints cannot be satisfied, e.g. no feasible schedule exists to complete every task by its deadline or after expending $8000 worth of labor, several task remain incomplete. If the DELS lacks authorization to violate the specified requirements as it does in the first scenario, then the controller must ask the source of the requirements, usually a supervisory controller or a customer, to relax a particular requirement, e.g. allow a due date to be extended or provide additional budget to complete the tasks.

However in many practical cases, the requirements may have some flexibility, e.g. there may be overtime labor available for an additional cost and tardy jobs may be accepted if a penalty is paid. Then the controller has the option, or *is authorized*, to incorporate these additional costs into the analysis model and then optimize the trade-off between using overtime labor and allowing jobs to be late and completed at a later time. When flexibility is authorized, requirements are extended to form goals, where there are penalties associated with violating a goal, but the DELS controller is authorized to do so (perhaps up to pre-specified level):

1. The system's total labor cost (weekly) goal is $ 8,000, but overtime is authorized at a cost of $60/hour.

2. The DELS shall set the goal to complete every by its due date, i.e., DELS.ontimeDelivery = 100%, but is authorized to pay a penalty of $100/day for late deliveries.

In value-driven decision making, preference structures in the form of marginal rate of substitution or trade-off between the total costs and on-time delivery (service-level) are formulated explicitly. For example, it may cost $15,000 to guarantee 100% on-time delivery, and the controller may be willing to spend $8000 to guarantee 95% on-time delivery. Then this trade-off is captured as the following relationship:

$$\text{totalCost} = 0.052 \exp\{12.57 * \text{ServiceLevel}\}$$

However by not explicitly accounting for late penalties, this approach does not fully incorporate the customers' preferences. These preferences may take the form of a separate utility function and require a negotiation approach or may be expressed by the customer as a schedule of prices and lead time over which the DELS can optimize.

4.2.1 Performance, Motivation, and Authorization

The examples discussed in the previous section integrated several different elements of decision making: performance definition, motivation, and authorization. This section will construct a formal definition of each of these elements, then the remainder of the broader discussion on decision-making (sections 4.2.2 - 4.2.4) will focus on formalizing the relationship between motivation and decision-support tools, where the motivation-types essentially provide templates for instantiating conforming analyses that are capable of answering 'motivation-driven' questions.

4.2.1.1 Performance Attributes

The performance attributes of the DELS are a specialized type of property (figure 36). The set of performance attributes can be refined to include specialized properties such as agility, reliability, responsiveness, cost, and asset management efficiency [244]. Using the stereotypes presented in figure 36, the performance attributes of a Warehouse can be defined (figure 37). In this example, the warehouse attributes are derived from the SCOR model [244], but these attributes can be further decomposed and may be accompanied by constraints that define how the performance attributes are calculated or measured. While a complete library of performance attributes is beyond the scope of this dissertation, a consistent set of metrics would simplify the design process by standardizing the information available.

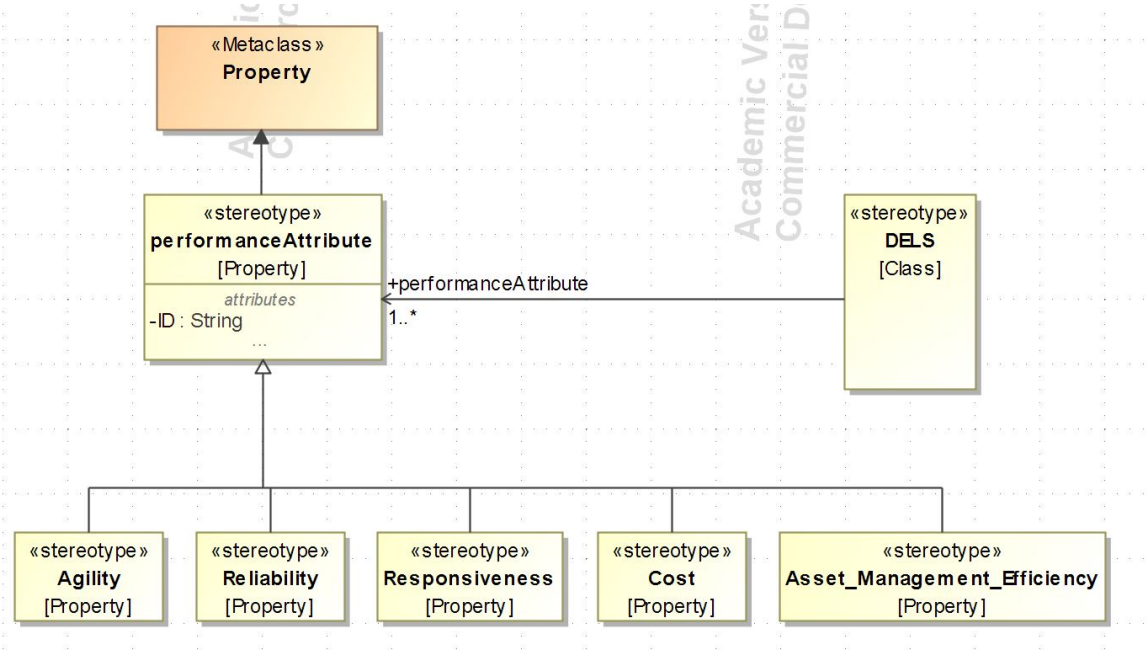


Figure 36: The performance attributes of the DELS extend and refine the properties of the DELS class

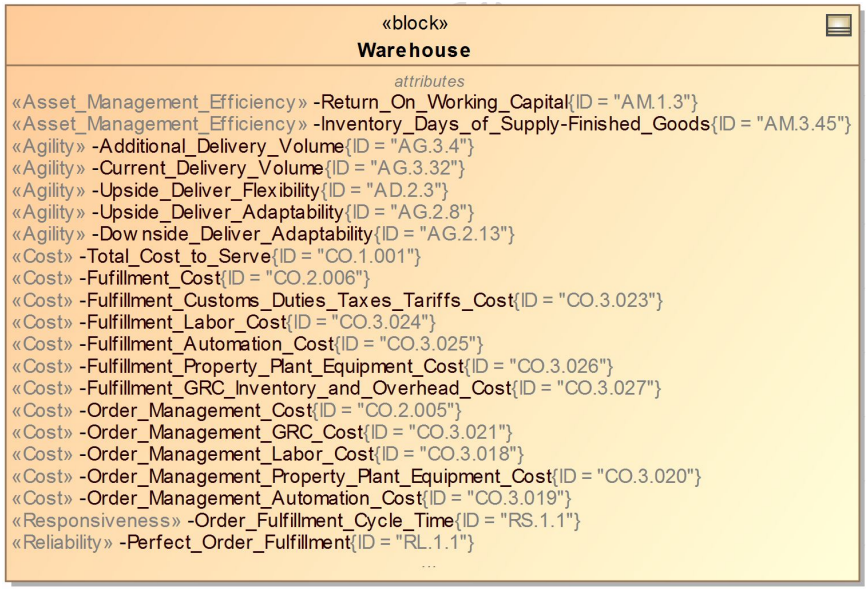


Figure 37: The performance attributes of the Warehouse use the performance attribute stereotypes defined in the DSL

4.2.1.2 Motivation

The Motivation of a DELS specifies how the DELS is expected to manage a particular performance attribute (figure 38). The motivation class defines a set of constraints captured

in the Object Constraint Language (OCL) [196] that can be used by the DELS decision-making process with respect to the motivation's *constrainedAttribute*. To capture knowledge about a particular property, OCL provides an operation, e.g. arithmetic or set-oriented; that manipulates or qualifies a property, and keywords, e.g. *if, then, else, and, or, not, implies*, that are used to specify conditional expressions. For each managed performance attribute, the motivation uses these constraints to define the *performanceDefinition* for that attribute.

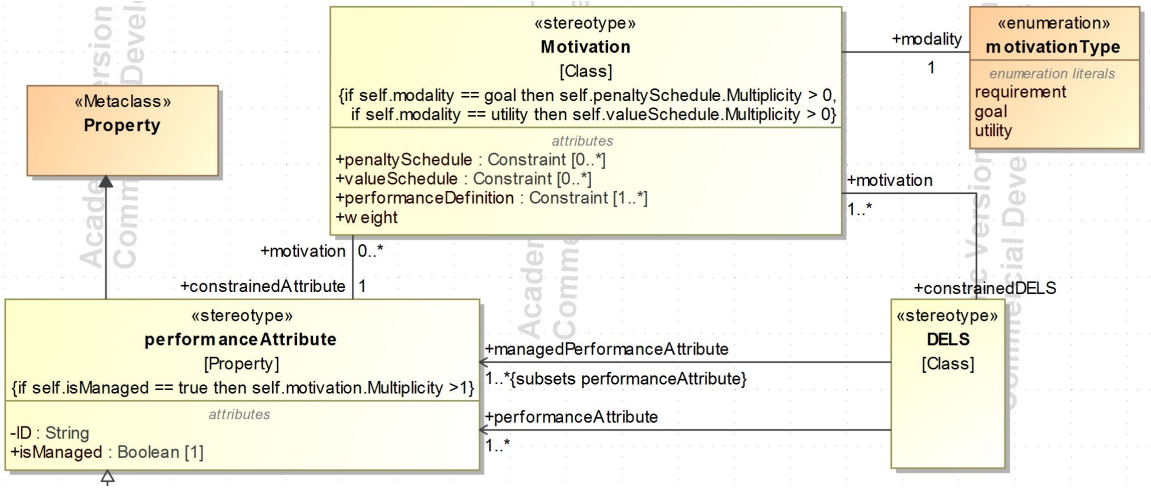


Figure 38: The motivation class of the DELS can be seamlessly updated and changed when needed.

Each motivation class has a *motivationType* that specifies whether the motivation is requirement, goal, or utility driven. If it is goal-driven, then a *penaltySchedule* must be specified, and if it is utility-driven, then a *valueSchedule* must be specified. The definition of these schedules will be discussed further in sections 4.2.3 and 4.2.4 respectively. Finally, there is a weight associated with the motivation which is used for balancing and prioritizing goals and utility functions when there are conflicts; for further discussion on multi-attribute decision making, see, e.g. [119, 278].

Multiple Motivation and Performance Definitions Each managed performance attribute is not constrained to have only one motivation to define the system's management of that attribute, which may incorporate differences in internal and external performance definitions or levels of soft goals and hard requirements. For example, there may be two

performance definitions for the same attribute, where the average daily throughput is specified as a soft goal, but the minimum weekly throughput cannot drop below a particular hard requirement. In this case, the performance attribute is defined using two different aggregation periods. Additionally, there may be a soft target goal governing labor costs with penalty scheduling reflecting overtime costs. However, overtime may only be authorized up to a particular level which is specified as a hard requirement on maximum labor costs. Another example from the VRP literature, "the classification scheme also allows the modeling of soft constraints, by means of penalty functions. As an example, consider a situation in which each driver can go up to a limited amount of overtime. This is modeled by combining a hard and a soft constraint on route duration: the hard constraint imposes a limit on overtime, and the soft constraint penalizes overtime." [77]

Additionally, there may be conflicting sets of goals over different planning horizons, such as long-term goals seeking to maximize average resource up-time (encouraging preventative maintenance), which may be in conflict with short-term goals, such as daily throughput targets. Attaining these goals over different periods is certainly a complex challenge for the controller, but when authorized, operational flexibility helps balance these conflicting goals. For example, if the daily throughput goal is between 90 and 110 tasks and it is hour 7 of an 8 hour shift and 105 tasks have been completed, then the controller may schedule preventative maintenance for the last hour. Likewise the controller may have to balance total maintenance costs against up-time, thereby having to take advantage of cheap maintenance during an inconvenient production period.

4.2.1.3 Authorization

In other domains, specifically security-oriented domains, authorization and authority are typically specified by yes/no access rules to data or methods [157, 257]; e.g. a user in this role can/cannot have access to those files. A formal definition of authorization policies as they relate to control decision-making will be presented in section 4.4.1 alongside policy-based management. Informally, authorization, as it is related to a controller's motivation, grants a controller the ability to violate a particular constraint, or requirement; possibly

specifying the conditions in which this authority is granted. This authorization must be granted by the entity that has jurisdiction over the requirement, and is specified as a part of the contractual agreement.

Authority in the context of operational control defines the difference between requirement-driven and goal-driven motivation. A requirement is defined as an inviolable constraint and the controller must seek specific authorization from the controller governing the requirement to relax the requirement. Goals, on the other hand, afford the controller flexibility in executing actions to attain a target performance metric by not requiring permission to violate the constraint but rather shaping the decision by specifying, a priori, the extent and penalties associated with violating the constraint. From the perspective of defining constraints on managed performance attributes, requirements are absolute constraints, violation of which rules out a potential solution, and goals are preference constraints that say which solutions are preferred [231].

4.2.2 Motivation: Requirements

In proposing the concept of satisficing and bounded rationality, Simon [256] conjectures that in today's complex organizations the decision makers do not try to maximize a well defined utility function but rather focus on satisfying objectives. Simon sought to explain the behavior of decision makers under circumstances in which an optimal solution cannot be determined: e.g. outcomes cannot be evaluated with sufficient precision, decision makers lack information about the relevant probabilities of outcomes, and they possess limited memory. Bounded rationality is the idea that when individuals make decisions, their rationality is limited by the information they have, the cognitive limitations of their minds, and the time available to make the decision. In this section, motivation in the form of requirements is defined as constraints that define operations that qualify an object's property, e.g. operational or performance attributes, and keywords that specify conditional and reactionary expressions. To make a distinction between types of motivation, these requirements are defined strictly without the authorization to violate the constraints. This form of motivation corresponds to formulating and solving constraint satisfaction problems.

Analysis: Constraint Satisfaction Problem Mathematically, a constraint satisfaction problem is defined in [231] as a triple $\langle X, D, C\rangle$, where $X = \{X_1, \ldots, X_n\}$ is a set of variables, $D = \{D_1, \ldots, D_n\}$ is a set of the respective domains of values (of each variable), and $C = \{C_1, \ldots, C_m\}$ is a set of constraints. Every *constraint* $C_j \in C$ is defined by a pair $\langle t_j, R_j\rangle$, where $t_j \subset X$ is a subset of k variables and R_j is a k-ary relation on the corresponding subset of domains. An *evaluation* of the variables X is a function from a subset of the variables to a particular set of values in the corresponding subset of domains. An evaluation v satisfies a constraint $\{t_j, R_j\}$ if the values assigned to the variables t_j satisfy the relation R_j. An evaluation is *consistent* if it does not violate any of the constraints. An evaluation is *complete* if it includes all variables. An evaluation is a *solution* if it is consistent and complete.

Performance requirements can originate from many internal and external sources and may come in many different flavors, e.g. (1) each task can provide its own set of requirements, (2) a supervisory controller can suggest performance targets, (3) the controller can also establish its own internal set of requirements to simplify the decision-making process, and (4) requirements can also form the basis of the decision rules, e.g. priority service rules. Since the controller is not authorized to violate the hard requirements, the penalty function value can be thought of as $+\infty$.

1. $\{finish(t) \leq t.dueDate\}, \quad \forall\, t \in TaskSet$
2. $\{\mathbb{E}[Revenue] \geq 2.5e6\}$ or $\{\mathbb{P}[Profit < 0] \leq 0.05\}$
3. $\{r.Utilization < 0.9\}, \quad \forall\, r \in ResourceSet$ or $\{WIP \leq 12\}$
4. $\{start(t_1) \leq start(t_2)\}, \forall\, t_1 \in \{tasks | task.class = 1\} \wedge t_2 \in \{task | task.class > 1\}$

Additionally, a domain specific constraint satisfaction language can be constructed by defining classes of constraints, e.g. precedence constraints (1) and resource sharing constraints (2) for the job scheduling problem [42].

1. $precede(t_1, t_2) \implies start(t_2) \geq start(t_1) + dur(t_1), \quad \forall t_1, t_2 \in T$

2. $resourceSet(t_1) = resourceSet(t_2) \implies start(t_2) \geq start(t_1) + dur(t_1) \vee start(t_1) \geq start(t_2) + dur(t_1)\ \forall t_1, t_2 \in T$

4.2.3 Motivation: Goals

If a DELS cannot violate hard, externally imposed, constraints such as task due dates, resolving infeasibility may require reneging on the contract or renegotiating the due date. This type of motivation may be overly restrictive and not allow an intelligent controller the flexibility required to balance the trade-offs among a set of key performance indicators. However, the concept of a constraint can be softened by adding a penalty schedule to the contract (figure 38), such as a simple, static late fee or a full schedule of penalties. This gives the controller more flexibility in balancing its commitments, but requires a greater degree of intelligence to plan and schedule the completion of the tasks.

Analysis: Goal Programming Goal Programming (GP) is a multi-objective analysis technique that supports decision-making [271]. The penalty schedule is incorporated into the objective function as under-target and over-target penalties, u_i and v_i respectively. Additionally, the relative importance of goals can be captured by simple weighted goals or lexicographical goal programming methods; the respective formulations are provided below:

$$\begin{aligned} \text{Weight} \ \min_{\mathbf{x} \in C_s} z &= \sum_{i=1}^{k} (u_i n_i + v_i p_i) \\ \text{s.t.} \quad f_i(\mathbf{x}) + n_i - p_i &= b_i, \ i = 1, \ldots, Q \end{aligned}$$

$$\begin{aligned} \text{Lex} \ \min_{\mathbf{x} \in C_s} \mathbf{a} &= (g_1(\mathbf{n}, \mathbf{p}), g_2(\mathbf{n}, \mathbf{p}), \ldots, g_L(\mathbf{n}, \mathbf{p})) \\ \text{s.t.} \quad f_i(\mathbf{x}) + n_i - p_i &= b_i, \ \ i = 1, \ldots, Q \\ \text{where } g_l(\mathbf{n}, \mathbf{p}) &= u_{l_1} n_1 + \cdots + u_{l_Q} n_Q + v_{l_1} p_1 + \cdots + v_{l_Q} p_Q \end{aligned}$$

Job Shop Example In this goal-driven job-shop scheduling example from [245], there are two goals that need to be considered when constructing a schedule: the makespan goal

and the flow-time goal. In each case, penalties are only assessed for exceeding the target goals, d_1^+ and d_2^+ respectively in the objective function.

$$\min_{\mathbf{X} \in C_s} Z = P_1 d_1^+ + P_2 d_2^+$$

Makespan Goal:

$$\sum_{j=1}^{N} A_M \times Z(j) + \sum_{j=1}^{N} X_j^M + d_1^- - d_1^+ = G_1$$

Flow-time Goal:

$$\sum_{j=1}^{N} (N - j + 1) \times (X_j^M + A_M \times Z(j)) + d_2^- - d_2^+ = G_2$$

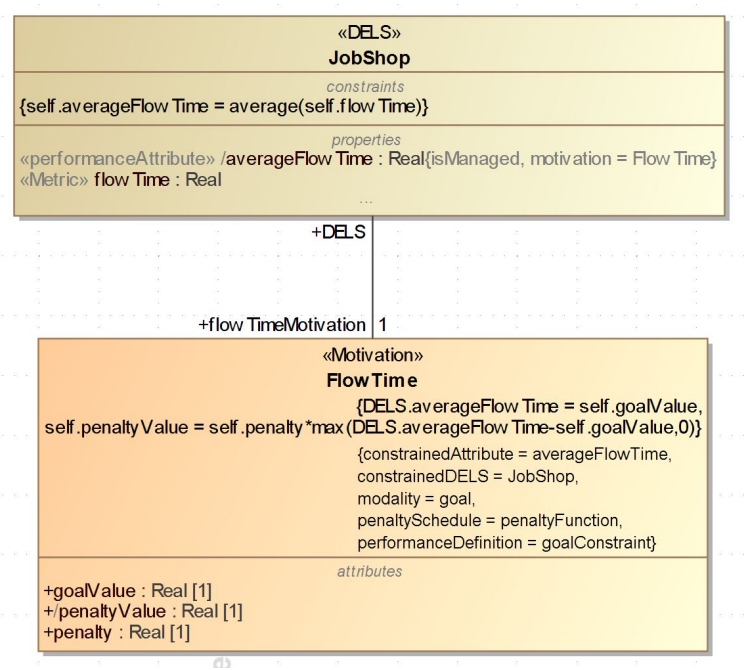


Figure 39: The motivation class for flow time designates the parameters, constraints, and functions governing the DELS evaluation of decision regarding its Flow Time metric

4.2.4 Motivation: Utility

In distributed decision-making frameworks such as agent-based systems, the autonomous agents seek to maximize their utility or value functions through a negotiation protocol that is typically governed by a variant of the contract net protocol [262]. Keeney and Raiffa [150] and Hazelrigg [129] describe the fundamental assumptions and formulations

that constitute multi-attribute utility theory (MAUT) and value-driven decision making. Whereas eliciting a multi-attribute utility function requires the designer to express his trade-offs over lotteries in the outcome space, the multi-attribute value function requires the designer, or stakeholder, to express the trade-offs he is willing to make between the various criteria: $X_1, \ldots, X_n$ [150, 129]. In each of these methods, the decision process requires eliciting the preferences of the stakeholders, where there may be many possibly conflicting objectives and uncertainty in the outcomes, and translating the preference structure into a mathematical statement that can be integrated into the controller's decision making process.

Analysis: MAUT Optimization Multi-attribute utility functions can have several different forms, but the following is a general formulation of a utility-based motivation for a set of performance attributes $x_1, x_2, \ldots, x_n$, where u_i is marginal utility function of attribute x_i and k_j is the weight of the utility argument j, where $r \geq n$ to allow for multiplicative utility effects [150]:

$$u(x_1, x_2, \ldots, x_n) = f[u_1(x_1), u_2(x_2), \ldots, u_n(x_n), k_1, k_2, \ldots, k_r]$$

A concrete example was given in the introduction of this section, but it has a more general form:

$$u(\text{laborCost}, \text{serviceLevel}) = k_1 u_1(\text{laborCost}) + k_2 u_2(\text{serviceLevel})$$

for some weights k_1, k_2 and marginal utility functions u_1, u_2

While agent-based methods are used to implement distributed architectures, multi-attribute utility approaches have limited applications; see, e.g., supplier selection and order allocation [235] and supply chain coordination [305]. While not explicitly formulating a utility function, Denkena et al. [74] suggest the relative weights of attributes may vary depending on the mission of a particular class of agent: "Different order agent types vary in divergent weights of goals within their utility functions: An order agent representing a rush order will rate the goal 'finish on schedule due date' higher than the goal 'using cost-efficient manufacturing processes', for example. An order agent representing a make-to-stock order should prefer the opposite goals."

> "Utility is the single issue that agents consider, and agents are assumed to be omniscient. Utility values for alternative outcomes are represented in a payoff matrix that is common knowledge to both parties in the negotiation. Each party reasons about and chooses the alternative that will maximize its utility. Despite the mathematical elegance of game theory, game theoretic models suffer from restrictive assumptions that limit their applicability to realistic problems. Real world negotiations are conducted under uncertainty, involve multiple criteria rather than a single utility dimension, the utilities of the agents are not common knowledge but are instead private, and the agents are not omniscient." [146]

Summary The motivation language definition discussed in this section provides guidelines, in the form of constraints, for a DELS to manage its portfolio of performance attributes. Motivation definitions, instantiated as objects, are interchangeable which provides a flexible mechanism to change and update motivation definition as the state of the system evolves. The class definition can implement an interface for querying and modifying a specific motivation object pertaining to a particular performance attribute. For example, if a superior controller wants to modify the weights of the controller's utility function, it can simply replace the existing motivation definition with a new one rather than attempting to modify the controller definition itself.

Furthermore, the set of constraints defined in the motivation classes can be mapped to the objective function or performance constraints in any analysis model that the controller may formulate to answer a control question related to the managed performance attribute. Since the DELS DSL defines the system and the control questions provide the decision variables, the objective function derived from the system's motivation is the last component to formulating a complete optimization analysis model. With the specification provided in an abstract manner, the design question is the following: How to set the aspiration levels? Or rather, where do the constraints, goals, and utility weights come from?

4.3 Formulating the Control Analysis Model: An Interface to Solvers

Formulating an analysis model to answer a particular instance of a control question should rely on a reusable and extensible mapping from the system model to an analysis model that is based on the abstract question definition and the DELS DSL. For each control question, there may be several analysis methods available to provide an answer, each with different solution quality and run-time performance guarantees. To answer a particular control question, the controller relies on the formulation component, which uses the context of the question and the system model to construct a particular answering analysis model.

However, it may be possible to construct and initiate multiple analyses concurrently and select the results with the best performance when a decision is required; e.g. select the best sequence of tasks identified after thirty seconds has elapsed. To take advantage of different analysis capabilities, multi-strategy systems integrate multiple inference types and computational representation mechanisms in one decision support system [186]. New technologies, such as multi-threading capabilities and web-based services, blur the line between high-level (distributed) and low-level (local) components and where the decision-making capabilities reside. Holonic architectures separate the high-level and low-level control layers [182, 296]. This bi-level architecture configures the controller with real-time sensing and decision-making capabilities (low-level), in the form of policies that are configured off-line. However, it also has high-level decision-making capabilities that run asynchronously in the background; e.g. the controller may construct an optimal schedule to use as a guide in future real-time decisions or as a template for repair heuristics [292].

A uniform interface to analysis models and their corresponding tools enables the formulation component of the controller to access a wide variety of solution methods for each control question. The idea of a uniform interface is embodied by the **strategy pattern** [102], which defines a family of algorithms, encapsulates each one, and makes them interchangeable (figure 40). The *AlgorithmInterface()* provides access to the optimization method and defines the required inputs and the expected output. Moreover, the Controller can define a *ContextInterface()* to provide a method for the Strategy object to access its data and the system model. This allows the algorithm to vary independently from clients

that use it; i.e., new analysis models and tools can be configured to work with the controller by properly structuring them to conform to the abstract AlgorithmInterface.

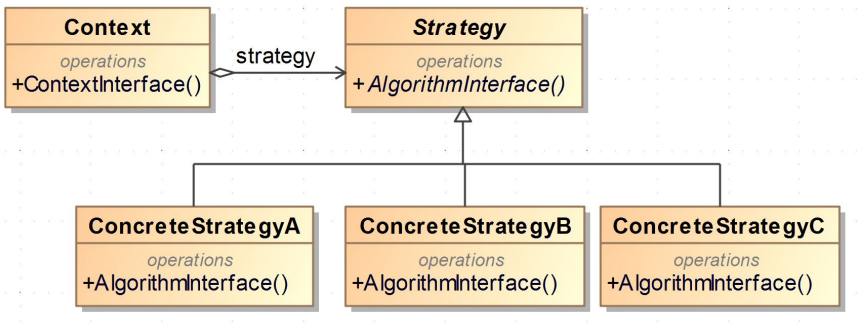


Figure 40: The strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable [102].

The formulation component is configured with an abstract strategy class for each control question to formulate the corresponding analysis models, where the target question defines the decision-variables, the motivation defines the objective function, and the system model provides the instance data, including the constraints associated with each task and resource. Each concrete strategy uses this information in different ways to construct different analysis models, but the uniform interface allows the details of this construction process to be encapsulated, or hidden, in the class's behavior definition. Providing this constructor functionality requires embedded capabilities such as aggregation, mapping, and estimating the system data as necessary, which is then passed to the corresponding analysis solution tool. This is an important step for leveraging the flexibility of interoperable analyses, because each analysis model operates on the system data at a prescribed level of aggregation, fidelity, and resolution which can be properly managed by the concrete classes. As a special case of constructing analysis models, the formulation function also is required to construct discrete event simulation models for the optimization function to utilize, when necessary. This section seeks to define an abstract strategy class for each control question defined in the previous chapter.

4.3.1 Admission Control: Which tasks get serviced?

The admission control strategy formulates an analysis model to determine whether a task should be accepted or admitted into the system. The abstract interface to this strategy, *admission()*, requires only the incomingTask (figure 41) as input, which contains the process plan, processing constraints, and the 'value' of the task.

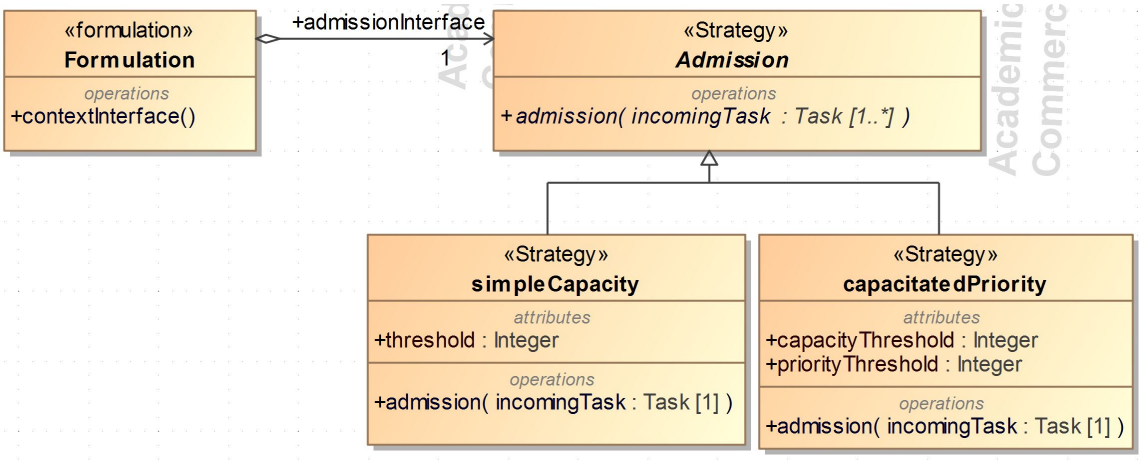


Figure 41: The abstract strategy classes can be sub-classed and refined to support many different solution algorithms.

The output of this analysis can be an admission token for the task, a boolean value that can be used by a gateway, or a callBehavior to an *admitTask()* function implemented in an actuator contained in the Plant (further discussion in section 4.5). The concrete strategy classes, *simpleCapacity* and *capacitatedPriority*, implement dynamic strategies as state machines (figure 42). By defining the rejection thresholds as properties of the strategy class, the parameterized strategy can be fine-tuned by a separate, off-line analysis model.

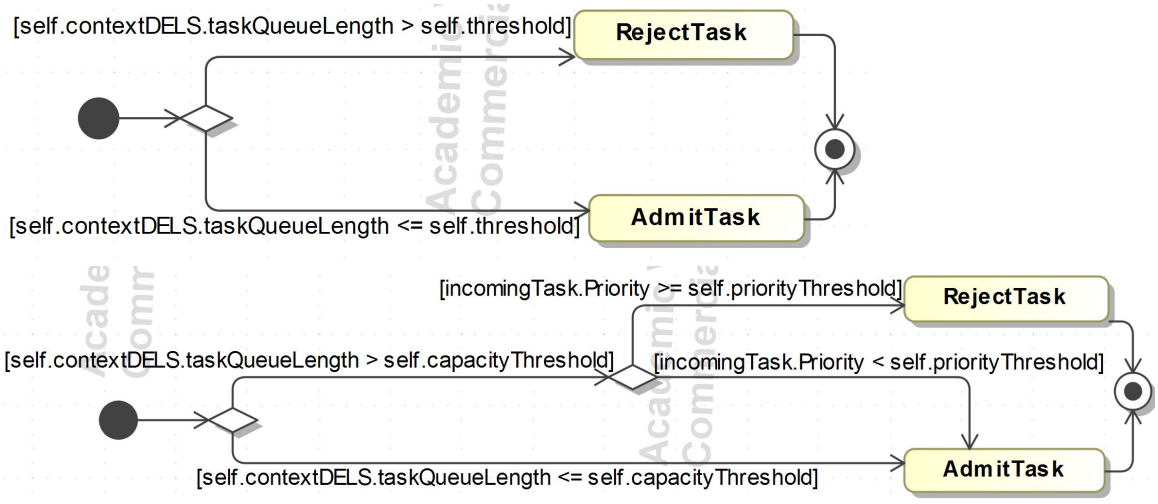


Figure 42: A state machine is one representation of the implementation of the simpleCapacity (top) and capacitatedPriority (bottom) admission strategies discussed in figure 41.

As mentioned above, the formulation function has the capability to formulate multiple analysis models to answer a specific control question. This capability allows the controller to use the same interface to construct static analysis models to guide the optimization of dynamic strategies. In this particular example, the controller has formulated a static admission control analysis model off-line or in the background that outputs the threshold policies that are implemented as the *simpleCapacity* and *capacityPriority* strategies which are used for dynamic on-line admission control.

4.3.2 Sequencing: When {sequence, time} does a task get serviced?

The sequencing strategy is responsible for indexing a set of tasks (figure 43). This sequenceIndex is used by the DELS' queueing mechanism to determine which task should be released next from the queue when the DELS is ready to process a new task. The sequencing can also be performed on orthogonal partitions of the taskList, thereby providing a complete sequence to all the tasks in the queue even if they may be partitioned or assigned to resources prior to sequencing.

The concrete sequencing strategy class in figure 43 simply sorts the tasks by how long the task has been waiting in the queue, *task.ageInQueue*; i.e. a FIFO sequencing rule (figure 43). The 'sort-by-attribute' strategy is a commonly implemented behavior by queues in

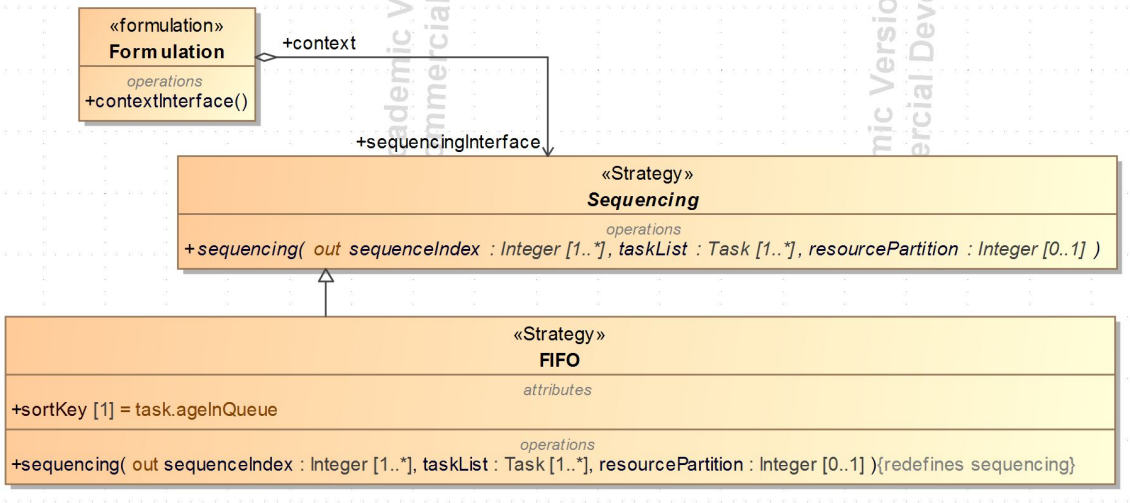


Figure 43: FIFO is a common sequencing strategy

simulation tools, but sequencing tasks by a sequenceIndex is more general mechanism that allows for an arbitrary sequencing solution that is the result of a complex or possibly black-box algorithm.

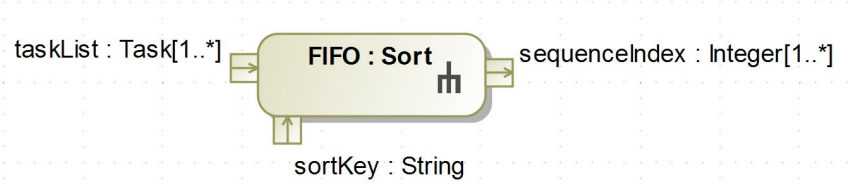


Figure 44: Sort-by-attribute functions are a common sequencing mechanism for discrete event simulation queues

4.3.3 Assignment: Which resource services the task?

Given a set of Tasks with defined capability requirements and Resources with capacity constraints, the assignment strategy produces a mapping of tasks to resources or partitioning of tasks among the resources (figure 45). Each task and resource contains a set of requirements and constraints that define the space of valid assignments, and which are accessible to the assignment strategy. The resourcePartition that is created by the strategy is a logical assignment, where each task that is assigned to a resource is logically, rather than physically, added to that resource's taskList. Furthermore, the assignment strategy may operate on a taskList that already has a defined sequence, or possibly a partial sequence, e.g. assigning

tasks to production periods rather than an complete ordering.

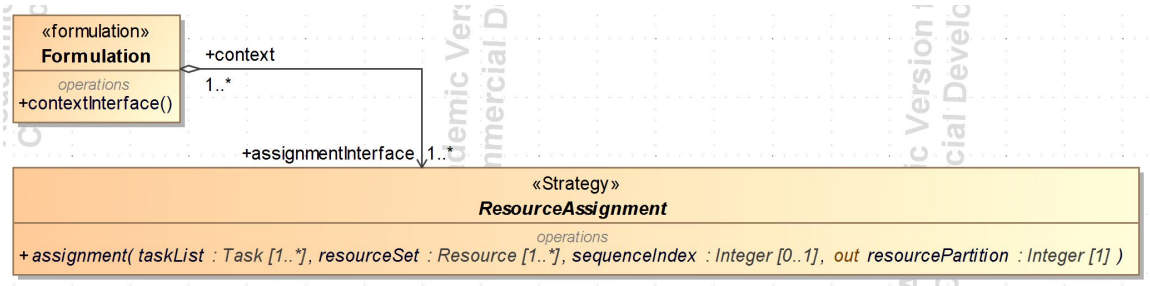


Figure 45: The ResourceAssignment strategy matches tasks to resources.

4.3.4 Routing: Where does a task go after service?

The routing strategy examines the process plan of the task and determines which process or path the task should take after exiting the system. In some cases, the routing strategy may only determine the *nextProcess*, and the corresponding DELS, to send the task to, but in other cases, the routing strategy may modify the task's process plan by pruning alternative processing plans or routes (figure 46). For example, some tasks may specify several alternative process plans captured as an intricate AND/OR digraph, and the routing strategy for the task's controller (cf. holonic control model [285]) at each step must prune OR branches or sequence AND branches to produce an executable process plan for the task.

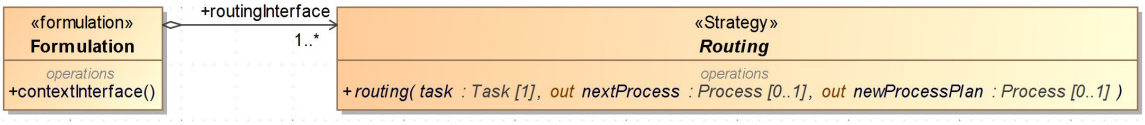


Figure 46: The routing strategy is responsible for selecting a complete process plan, or just the next process for the task.

The extent to which the routing strategy can modify or select alternative process plans is governed by the controller's authority over the task and the processes specified in the task's process plan. In centralized control architectures, the controller has global information and authority and decides the complete routing for each task and resource. But in local decentralized control architectures, the local controller may only be able to determine the process plan for the local process it has been assigned, and the next DELS that the task must

visit after leaving its domain. In holonic architectures, the task is continuously negotiating and re-configuring its process plan.

4.3.5 Resource State: Change the state of a resource?

The resource state strategy is the catch-all strategy for determining when a resource needs to change its state and to which state it should change. The strategy examines a particular resource and determines which *newState* it should transition to and optionally generates an *overheadTask* to execute the state change (figure 47).

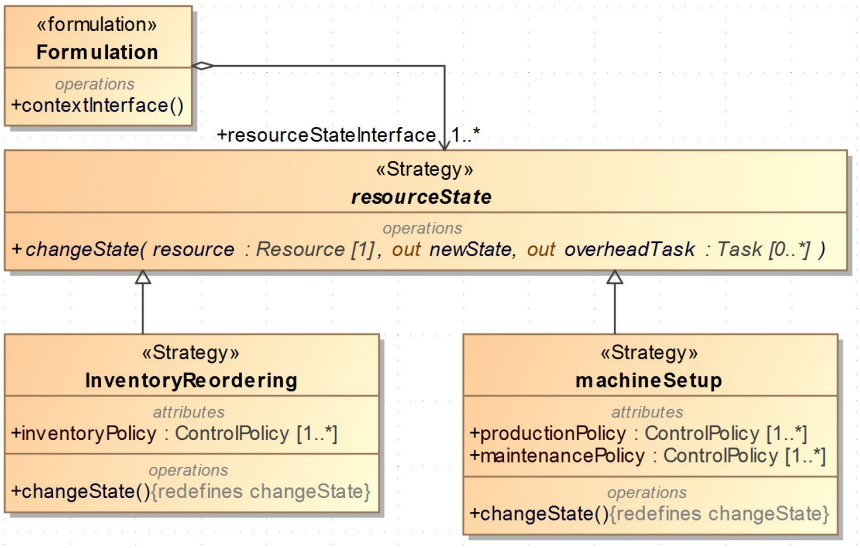


Figure 47: The resourceState strategy is responsible for determining when and to which state a resource should change.

The *inventoryReordering* and the *machineSetup* strategies (figure 47) have been selected to highlight that the abstract strategy and interface definition can be applied to both capacitated resources, where the strategy uses an inventory policy to determine when to reorder and how much to reorder, and discrete-state resources, where the production and maintenance policies determine when to schedule maintenance and which product or process the resource should be set-up to execute.

4.3.6 Joint Strategy Classes: Scheduling Example

Analysis models can be formulated to address two or more atomic control questions at once; see, e.g. section 3.6. This section elaborates the joint problem class for scheduling

with a corresponding strategy class for formulating an analysis model (figure 48). In the joint strategy class, the scheduling interface implements both the ResourceAssignment and Sequencing interfaces. This implies that a schedule is created each time the sequencing or assignment interfaces are invoked, and returns the sequenceIndex or resourcePartition, respectively. For expositional clarity, the signature of the operations in the abstract strategy classes sequencing and assignment are hidden in figure 48, but are the same as the definitions in figures 43 and 45, respectively. The redefined assignment and sequencing operations in the scheduling strategy class provide data access methods, i.e. the scheduling algorithm produces a schedule and then the *sequencing()* method examines the resulting schedule and returns the sequenceIndex for the set of specified tasks.

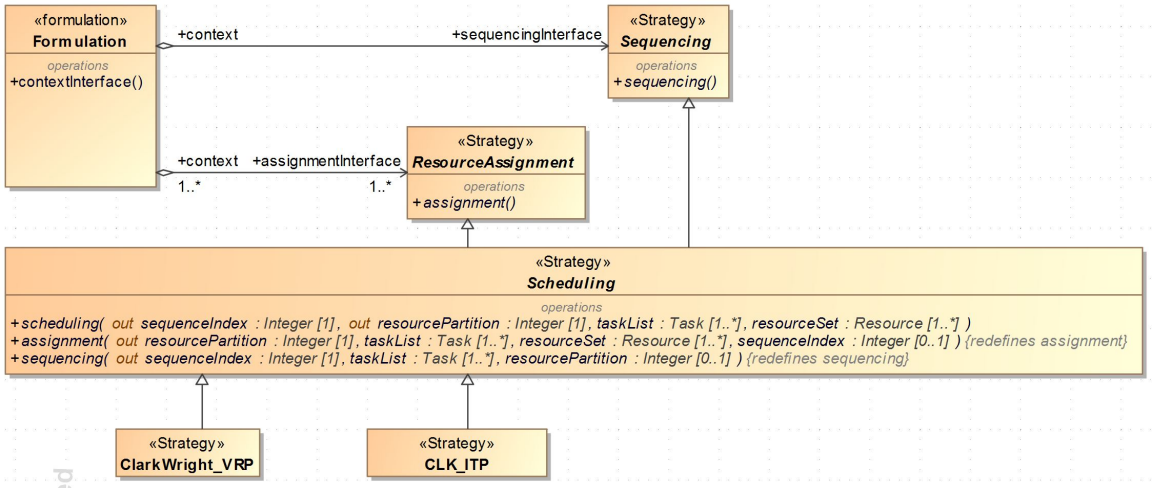
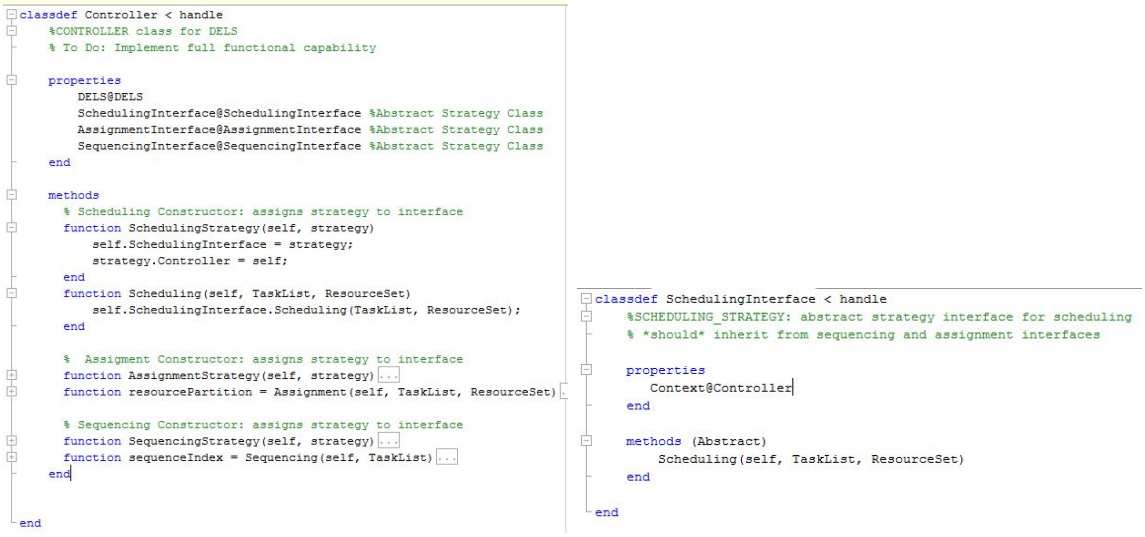


Figure 48: The scheduling strategy implements both sequencing and resource assignment interfaces.

The remainder of this section constructs a warehouse picking scheduling use case that demonstrates the application of the joint scheduling strategy where multiple scheduling strategies are implemented to provide the same analysis capability. In this example, the controller is configured with an abstract SchedulingInterface that implements the *scheduling*(TaskList, ResourceSet) interface (figure 49). Then the two specialized algorithms, Clarke-Wright savings algorithms [57] and iterative tour partitioning (ITP) [124], are constructed as concrete subclasses that conform to the interface specified the scheduling interface (figure 50).



```
classdef Controller < handle
    %CONTROLLER class for DELS
    % To Do: Implement full functional capability

    properties
        DELS@DELS
        SchedulingInterface@SchedulingInterface %Abstract Strategy Class
        AssignmentInterface@AssignmentInterface %Abstract Strategy Class
        SequencingInterface@SequencingInterface %Abstract Strategy Class
    end

    methods
      % Scheduling Constructor: assigns strategy to interface
      function SchedulingStrategy(self, strategy)
          self.SchedulingInterface = strategy;
          strategy.Controller = self;
      end
      function Scheduling(self, TaskList, ResourceSet)
          self.SchedulingInterface.Scheduling(TaskList, ResourceSet);
      end

      %  Assigment Constructor: assigns strategy to interface
      function AssignmentStrategy(self, strategy) ...
      function resourcePartition = Assignment(self, TaskList, ResourceSet) ...

      % Sequencing Constructor: assigns strategy to interface
      function SequencingStrategy(self, strategy) ...
      function sequenceIndex = Sequencing(self, TaskList) ...
    end


end
```

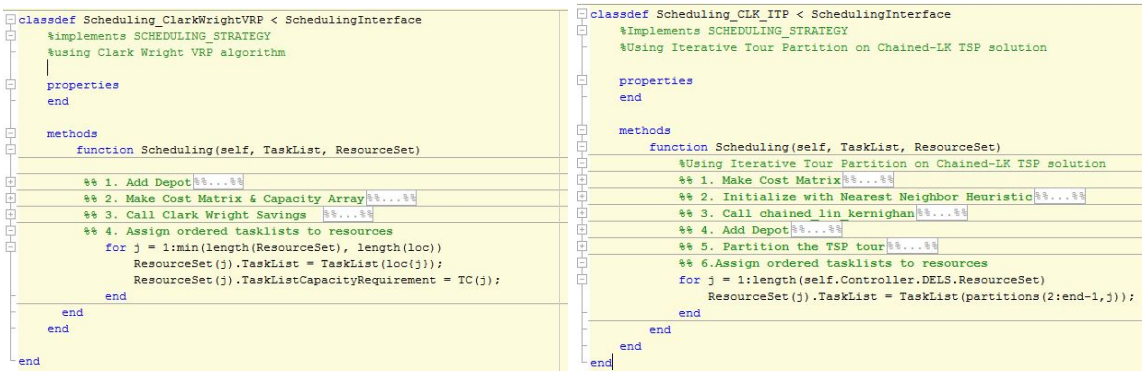
```
classdef SchedulingInterface < handle
    %SCHEDULING_STRATEGY: abstract strategy interface for scheduling
    % *should* inherit from sequencing and assignment interfaces

    properties
       Context@Controller
    end

    methods (Abstract)
        Scheduling(self, TaskList, ResourceSet)
    end

end
```

Figure 49: The controller (left) and the abstractscheduling strategy class (Right)



```
classdef Scheduling_ClarkWrightVRP < SchedulingInterface
    %implements SCHEDULING_STRATEGY
    %using Clark Wright VRP algorithm

    properties
    end

    methods
        function Scheduling(self, TaskList, ResourceSet)

         %% 1. Add Depot %%...%%
         %% 2. Make Cost Matrix & Capacity Array %%...%%
         %% 3. Call Clark Wright Savings  %%...%%
         %% 4. Assign ordered tasklists to resources
            for j = 1:min(length(ResourceSet), length(loc))
                ResourceSet(j).TaskList = TaskList(loc{j});
                ResourceSet(j).TaskListCapacityRequirement = TC(j);
            end
        end
    end

end
```

```
classdef Scheduling_CLK_ITP < SchedulingInterface
    %Implements SCHEDULING_STRATEGY
    %Using Iterative Tour Partition on Chained-LK TSP solution

    properties
    end

    methods
        function Scheduling(self, TaskList, ResourceSet)
            %Using Iterative Tour Partition on Chained-LK TSP solution
            %% 1. Make Cost Matrix %%...%%
            %% 2. Initialize with Nearest Neighbor Heuristic %%...%%
            %% 3. Call chained_lin_kernighan %%...%%
            %% 4. Add Depot %%...%%
            %% 5. Partition the TSP tour %%...%%
            %% 6.Assign ordered tasklists to resources
            for j = 1:length(self.Controller.DELS.ResourceSet)
                ResourceSet(j).TaskList = TaskList(partitions(2:end-1,j));
            end
        end
    end
end
```

Figure 50: The abstract scheduling strategy is implemented using Clark Wright for VRP (left) and Chained Lin-Kernighan with Iterated Tour Partitioning (Right)

In each case, the details of the scheduling algorithm are encapsulated in the strategy class, and since the class conforms to the scheduling interface the two strategies can be interchanged seamlessly. This encapsulation process can be done by constructing the analysis tool to conform to the abstract interface using the DELS DSL. It can also be done ex-post, which is more common. In this case, the existing tool must be "wrapped" in a conforming class definition and the interface must be mapped to the tool's existing interface. In figure 50, the second approach was used which manually maps the existing interfaces to the DELS DSL and abstract scheduling strategy interface and then calls the existing algorithms. With a stable DELS DSL, this a one-time investment to provide plug-and-play functionality and integrate a particular tool into the DELS analysis library.

Summary The formulation component is functionally responsible for constructing an analysis model to answer a particular control question about the system. There may be many analysis models that are capable of answering a particular class of control questions, and there may not be a best algorithm for answering each control question even within a limited scope of system states. The strategy pattern is discussed in this section to meet the broad requirements of bridging from the system model to an analysis model(s), by encapsulating each algorithm and providing a uniform interface to accessing the algorithm. Using this method, analysis tools can be constructed to conform to a plug-and-play ecosystem, and the controller simply needs to select an algorithm to answer the question at hand.

4.4 Implementing Analysis Results: Rule- and Policy-based Management

Formulating an analysis model that can be shared with a solver typically requires translating the problem into mathematical syntax, thereby stripping any domain knowledge from the problem. However, then the answer to the control question is provided in the same semantic-free abstraction. Therefore, there remains a need for a formal specification of the output of optimization solvers that adequately expresses the optimal control decisions to be executed.

Policies have a nice colloquial meaning in the operations research literature, but in artificial intelligence applications, a policy is usually defined as a complete mapping from states (of the world) to actions [231]. In policy-based management systems, a formal language implementation provides controllers with a mechanism to construct and share policies with other controllers. Furthermore, policies are treated as objects that provide operations for querying and modifying policies which enables policy-level execution flexibility and agility to constantly evolving system knowledge and objectives [257, 154]. Finite state machines and production rule systems are capable of organizing these rules and providing an execution mechanism to decide when and which rules to activate.

In policy-based management methods, an intelligent controller is configured with an inference and reasoning engine that operates on the rules and priorities. These rules and priorities can be configured off-line and given to the controller to implement, but a uniform execution mechanism does not require re-programming the controller. Separating the policy from the implementation of a system permits the policy to be modified in order to dynamically change the strategy for managing the system and hence modifying the behavior of a system, without changing its underlying implementation [257]. Additionally, rule-based systems have the advantage of being reactive, capable of responding in real-time.

The result is an explicit mechanism for implementing the output of analysis tools by first restoring the semantic content to the analysis solution and transforming that solution into formal rules, which then can be executed using a finite state machine, production rule system, or a comparable tool. This is an important requirement for interoperability between analysis tools and mechanisms to execute control actions in the system.

This section will first define rules and policies and then discuss policy-based management in the context of DELS (section 4.4.2). The second half of this section will be focused on the formal relationship between policy-based management and execution mechanisms (section 4.4.2).

4.4.1 Rules and Policy-based Management

Policy objects exist informally in the DELS literature to describe the protocols used in procurement, manufacturing, transportation, and distribution of material within the supply chain. From the Markov decision process (MDP) literature [214], the analysis output is a policy structured as follows: given a state of the system the decision maker will choose a specific action from a defined action set. The set of valid actions is defined by the class of control question, and the policy space consists of all sequences of control actions admissible at different states [44]. In rule-oriented holonic control models, a rule as composed of a condition and a corresponding action [254], where the condition is evaluated with respect to an attribute or the state of a resource and the corresponding action instigates a method of the resource to change its state.

Research on policy-based management of networks and distributed systems provides some formalization of these basic definitions [157, 257, 62]. Koch et al. [157] define a core set of attributes for describing a policy including the relationship between the managing and managed objects, the action to be executed, any conditions that must be fulfilled prior to execution, and triggering events (figure 51). Damianou et al. [62] expand on the different types of constraints: subject/target state, action/event parameters, and time constraints. Sloman [257] defines a policy constraint as a predicate that refers to global attributes, such as time, or action parameters, e.g., temporal constraints (policy applies before/after/between), parameter value constraints, and preconditions. Policies also can be defined as authorization policies which define what a controller is *permitted* or *not permitted* to do, or obligation policies, which define actions that a controller *must* or *must not* take [257]. Sloman [257] proposes utility as a method to manage high-level policies, where a utility function $u(S)$ is a function of the service attributes S.

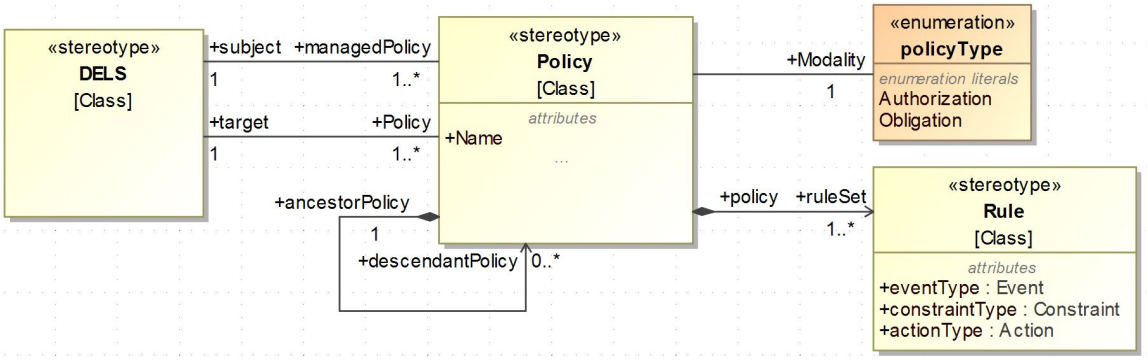


Figure 51: Policy Definition Language [157]

The Policy Definition Language enables a computer to check the syntax of a given policy description and translate policies into executable rules [157]. The Policy Description Language uses the event-condition-action rule (E[C]A) paradigm of active databases to define a policy as a function that maps a series of events into a set of actions [175]. The language can be described as a real-time specialized production rule system used to define policies. Reaction RuleML formalizes production rules (condition/action), E[C]A rules, and complex rules: "Reaction RuleML allows for standardized rule interchange, semantic interpretation and translation, and distributed event- messaging interactions in loosely-coupled and decoupled distributed rule-based systems such as Web inference services and semantic agents." [204]. OMG has also proposed a Production Rule Representation standard for production rules as an extension to OCL [212].

While a policy is meant to govern the behavior in a relationship between two DELS (figure 51), rules are the fundamental semantic objects to represent system knowledge. E[C]A rules are a particular type of rule that specifies the relationship between the behavior and state of the system. A policy uses a rule to specify required behaviors in reaction to specific states or events while incorporating context-specific knowledge about the relationship.

4.4.1.1 Formal Relationship Between Policies and Finite State Machines

The controller needs a formal specification of the execution behavior of the policies and rules. Policies consisting of rules can be formally mapped to finite state machines (FSM), which provide a formal specification for *implementing* the prescriptive analysis results; that

is, translating the output of optimization module into control code [215].

For each rule, the premise clause, antecedent clause, or condition is a Boolean combination of predicate functions on associative triples, each having four components: a predicate function, an object, an attribute, and a value [64]. By definition, the predicate is a Boolean-valued function $P : X \to \{\text{true}, \text{false}\}$, called the predicate on X. One way to interpret the implementation of these rule-systems is as a hierarchy or ordered set of if-then clauses [2].

More generally, there is an equivalence between FSM and production rules; i.e. if you can express a rule as a regular expression, you can design a finite-state machine that carries out the rule [274]. Furthermore the E[C]A rule formalism, defined as $\{Events\}\ x\ \{Conditions\ \} \to \{Actions\}$, where the conditions are derived from the state of the system and the events are input from external stimuli, is equivalent to the transition definition in a Mealy machine [185] $T : S \times \Sigma \to S \times \Lambda$ (This is actually the combined action/state change formulation). Whereas the output of Mealy machines is determined by its current state and current inputs (outputs on transition edges), the Moore machine outputs are determined by its current state through entry and exit actions [191]. The UML finite state machine formalism integrates characteristics from both Mealy machines and Moore machines, and provides modeling capabilities for hierarchically nested states and orthogonal regions. The orthogonal regions coupled with asynchronous execution behavior allows FSM formalism to provide a flexible and reactive execution mechanism for rules in DELS controllers.

4.4.1.2 Production Rule Systems

Production-rule systems (PRS) extend the execution behavior of a finite-state machine with a reasoning component that clarifies the execution behavior when there may be many rules that are enabled at once. A production rule system is an un-ordered set of relatively independent data sensitive rules coupled with engines for executing production rules, providing inference, prioritization, and deterministic repeatability [64, 286, 5, 17].

In production rule systems, the rule interpreter, or inference engine, first matches production rules against its model of the system, and then selects which of the matched rules to execute. They traditionally have to choose often between mutually exclusive productions

since actions take time, only one action can be taken, though this may not be the case in loosely coupled systems with asynchronous behavior execution. The PRS is configured with a strategy to cope with simultaneous matched rules, which may vary from simple lexicographical, weighted, or priority strategies to complex strategies such as sorting based on the times at which each production rule was previously fired or the expected resulting modifications to the system. PRS is a more general way to implement and execute rules and policies and has been demonstrated as a viable knowledge representation formalism for intelligent FMS controllers [251, 143].

Production rule systems rely on matching the condition of the rule to the state of the system, and state changes cause a review of the production rules. However in E[C]A rule systems, an event triggers the review of the corresponding collection of rules, which is in agreement with the event-driven management paradigm discussed in the monitoring section. In [253], rather then having the PRS monitor or search the system for state changes, the inference process is initiated by system notifications, thereby producing a rule-based system similar to E[C]A rules.

4.4.1.3 Designing Rule-based Systems

Designing a rule-based system traditionally requires eliciting rules from domain experts. Alternatively, operations research techniques can be used to design rule-based systems, and various architectures consider operations research analysis models and solutions tool as part of their structure (tandem expert systems) [163], e.g. the action resulting from a rule could call for an optimal scheduling algorithm, and the resulting schedule could be used as a guide for future real-time decision making.

While many rule-based systems, especially expert systems, have the ability to explain the reasoning process or inductive path used to arrive at a particular recommendation, artificial neural networks (ANN) use learning algorithms to deduce the functional relationship between inputs and outputs. In the context of decision-making, these methods obscure the actual pathway that maps events to actions. However, rules can be a direct output of the decision support or can be harvested or learned from the analysis, e.g. neural networks

[307, 9, 2]. Mouelhi-Chibani and Pierreval [193] use simulation-optimization methods to train a neural network to select appropriate control rules.

4.4.2 Applying Policy-based Management to DELS

A policy is an abstract definition with many valid policy structures and probably an infinite number of policy instances. Typically, optimal-control analysis models seek a policy for a particular use case, or the expected case; e.g. define the optimal WIP threshold for the controller to reject new incoming tasks. In practice though, there may be many rules governing a wide range of states, scenarios, and use cases, but tractibility issues limit any single analysis' ability to formulate a complete collection of rules to govern every state of the system. Rather than discuss specific policies from the DELS literature, this section first focuses on the how the literature defines the structure of policies, defines the triggering events, and evaluates the quality of a policy. Then this section applies policies to provide the real-time reaction-capability to the monitoring function and the role that authorization and obligation policies have in enforcing contracts.

There is often a dynamic, temporal component to decision making in DELS, and policies need to specify sets of actions that take place over a period of time. Dean et al. [69] separate policies into non-adaptive policies that designate a priori a fixed sequence of tasks to service or adaptive policies that make dynamic choices based on the (instantiated) resource consumption of the preceding tasks. McGavin et al. [183] create a continuum between adaptive and non-adaptive policies, stating that an allocation policy is characterized by four decisions: number of withdrawals from stock, times between successive withdrawals, quantity of stock to be withdrawn, division of withdrawn stock among the retailers. These characteristics can be generalized to: number of times during a period to make decisions on assigning tasks to resources, the length of the inter-decision period, how much of the resource's capacity to allocate at each decision point, and how to divide up the capacity among task/customer classes.

In many applications, the triggering event often is defined implicitly. For example, in inventory management systems that use a base-stock policy, there exists either a critical

inventory position S that triggers a *notification event* or a periodic inventory review, a *polling event*. Then if the inventory position is below S (*condition*), it is optimal to produce (*action*) [120]. Therefore, in continuous review systems, the inventory position would notify the controller once it reaches its critical value, but in periodic review systems, the controller must poll the level at periodic intervals.

However, in monitoring condition-based degradation signals, Wu et al. [303] specify a well-defined event for initiating a maintenance activity, including a polling frequency, filter type, presumably an interface to the signal acquisition board, and a threshold that triggers the maintenance event. McGavin et al. [183] integrate an explicit event definition into the policy specification by providing both the events, time-based or polling, and additional parameters for selecting or configuring a scheduling algorithm; e.g. how often and how far into the future to schedule. In a rescheduling policy, the additional attributes of the policy include schedule stability, schedule nervousness, and schedule robustness [292].

4.4.2.1 Revisiting the Monitoring Function

The controller's monitoring function is configured with policies that react to notification events about changes in state of the tasks and resources in its plant, e.g. PROSA, CAMX, and CMSD define broad classes of events related to the flow of tasks through a system and the resources executing the required processes, including failures, availability, maintenance, completions, arrivals [285, 73, 225]. Each type of incoming messages is associated with a production rule [143].

Additionally, polling events are initiated by the monitoring function to update and maintain its internal working model of the system. Ingham et al. [144] describe knowledge goals that express constraints on desired nominal values of state variables; e.g., 'Camera temperature standard deviation is less than 0.5 degree Celsius from 1:00 pm until 5:00 pm,' or 'Camera power switch position is known with 95% certainty or better from 1:00 pm until 5:00 pm.'. These knowledge goals are embedded into a policy governing when and how the monitoring process should go about updating the system model, which then queries other 'local' controllers and incorporates their feedback into the system model.

To understand and synthesize the feedback, the monitoring function uses descriptive and predictive analysis methods to formulate an expected trajectory of the system; i.e. "given the state of my system, what is a reasonable course for this system to take". This expectation, or rather excursions from the expectation, forms the basis for many of the internally-defined events that control policies and rules use to specify thresholds for taking action; i.e. can you predict that a task is going to be late or if the system will become overloaded. In this simple example (figure 52), the controller simulates the expected cycle time of tasks (top curve) and WIP levels (bottom curve) to predict the impact of a break-down event and then uses that predictive analysis coupled with a control policy to determine if or when to stop admitting tasks to the system. In this example, the upper control limits, 50 minutes and 20 units of WIP respectively, define the conditions that would trigger a response from the controller, such as rejecting any incoming tasks.

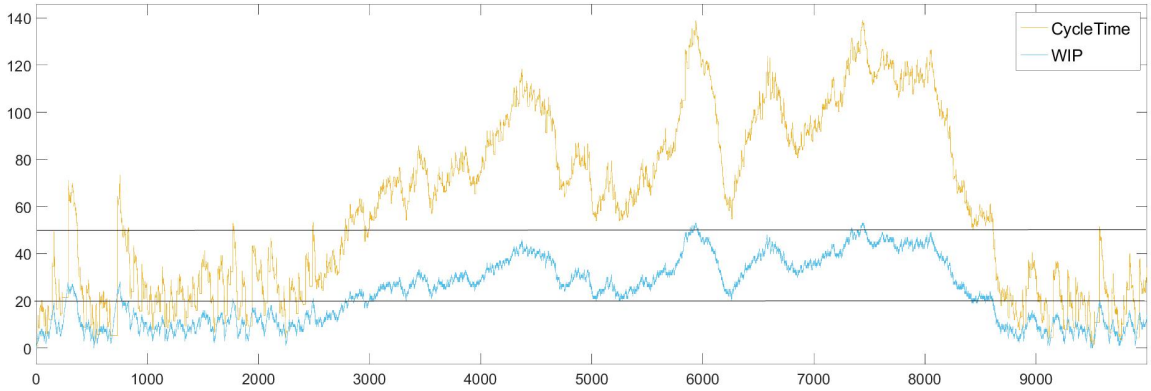


Figure 52: The monitoring function uses descriptive and predictive analysis models to produce an expected trajectory of the system. In this "Control Chart" control approach, the cycle time (top curve) and WIP level (bottom curve) are plotted with their upper control limits, 50 minutes and 20 units respectively, which trigger control policies.

4.4.2.2 Revisiting Motivation and Authorization Policies

Policies may be used as a way to implement and enforce externally specified requirements and goals, such as those resulting from a contractual obligation (figure 53). The contract contains one or more policies which specify the rules for executing the contract and governing the relationship between the service requester and the service provider. These rules are then configured into the controller's execution mechanism.

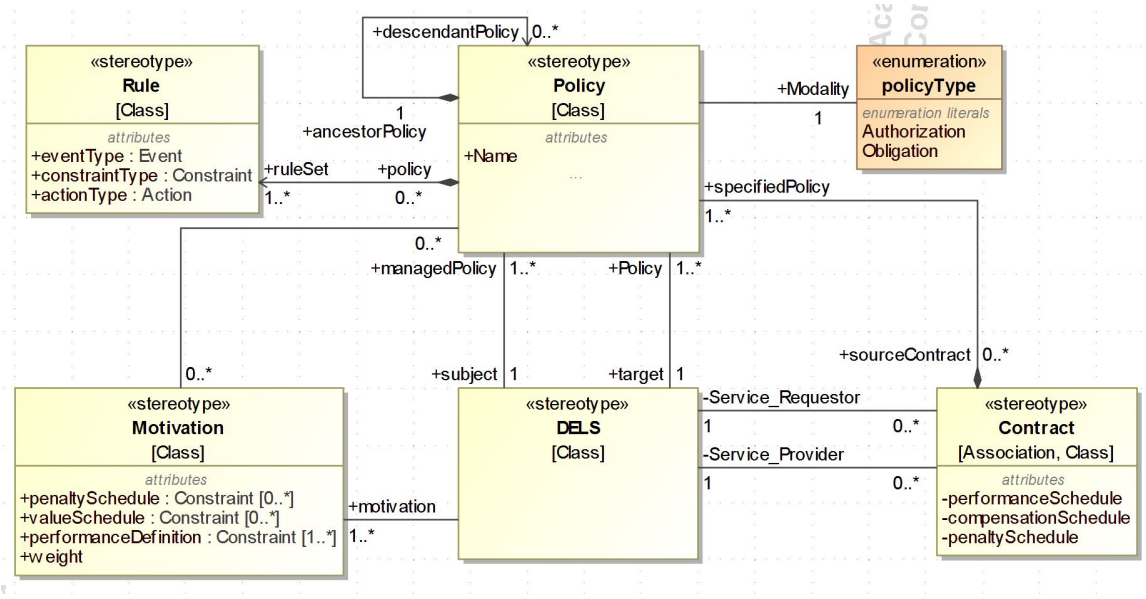


Figure 53: The authorization policy defines in which states a DELS can execute a particular behavior or violate a motivation constraint; e.g. in what circumstances can the controller renege on a task?

There may be several rules contained within a policy that governs a single entity within a controller's domain. The contract may specify that degree of authorization that the controller can exercise, such as violating due date or cost constraints, but also may specify a set of obligation policies, such as notifying the service requester of variances from the contractual agreement.

For example, consider the following contract that specifies a set of policies governing the acceptable behavior for late completion (figure 54). If the task is going to be < 1 day late, the controller is obligated to *notify()* the service requester. If the task is going to be < 7 days late, the controller may be authorized to *proposeNewDueDate()* or is obligated to *requestNewDueDate()*. Finally, if the task is going to be late by ≥ 7 days, the serviceProvider may be obligated to *renege()* on the task.

Furthermore, authorization policies may be useful for distributed and mediator type architectures by defining which controllers can listen and react to which events/resources/-subordinate controllers or may govern which types of resources can join a particular class of controller objects. "Authorization policies define what activities a member of the subject domain can perform on the set of objects in the target domain. These are essentially

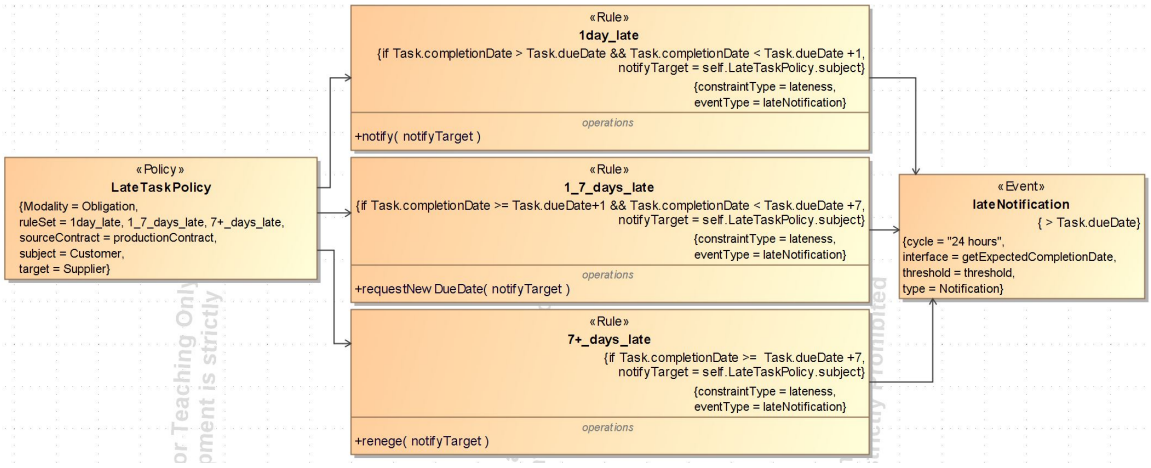


Figure 54: The obligation policy states the lateness threshold at which the service provider must notify, requestNewDueDate, and renege on the contract.

access control policies, to protect resources and services from unauthorized access" [62]. Authorization policies are also necessary for specifying the exposed operations required for conducting polling events, see section 4.1.3

Policies are a flexible way of specifying authorization and obligation for both cases where the DELS has it's performanceAttributes monitored by a supervisor and cases where the performanceAttributes for tasks and resources are governed by a distributed, contractual mechanism.

4.4.2.3 Discussion: Plans vs. Policies

In section 2.2, process plans were discussed as the primary language for a complete, static specification of the behavior of a DELS, which suggests an inherent conflict between the usage of plans versus real-time rules. However, process plans and rules are inter-related but require the specification of conditions for the controller to switch between static and dynamic control methods. This leads to a deeper discussion that is beyond the scope of this dissertation, but focuses on evaluating the trade-off between the response-time of dynamic policies and the quality of more time-consuming strategies.

Notionally, rule-based systems are used for online, real-time decision making, but these rules can be extrapolated to create complete process plans or schedules; i.e. there exists a dynamic sequencing rule that only selects the next task to be serviced, but the rule can be

applied recursively on the remaining tasks to produce a complete process plan. Policies may also give rules to traverse an existing process plan, or specify conditions when an optimal, but technically infeasible, schedule can be repaired heuristically and then used to determine the next task to service [292].

4.5 Executing Analysis Results: Call Behavior Specification and Plant Actuators

While the implementation function restores the semantics to the output of the analysis tools, there is a need for the functional specification of the plant behavior and the mechanism that executes the control actions prescribed by the controller. A uniform execution mechanism is essential to constructing uniform DELS controllers that are capable of integrating several system models from non-homogeneous domains. Finite-state machines provide an execution formalism for the controller to execute rules, and this section develops the functional specification of the plant's execution behavior consisting of a computational rule (which provides the abstract specification of the execution behavior), an actuator that implements the execution behavior, and a callBehavior that the controller uses to invoke the actuator's behavior.

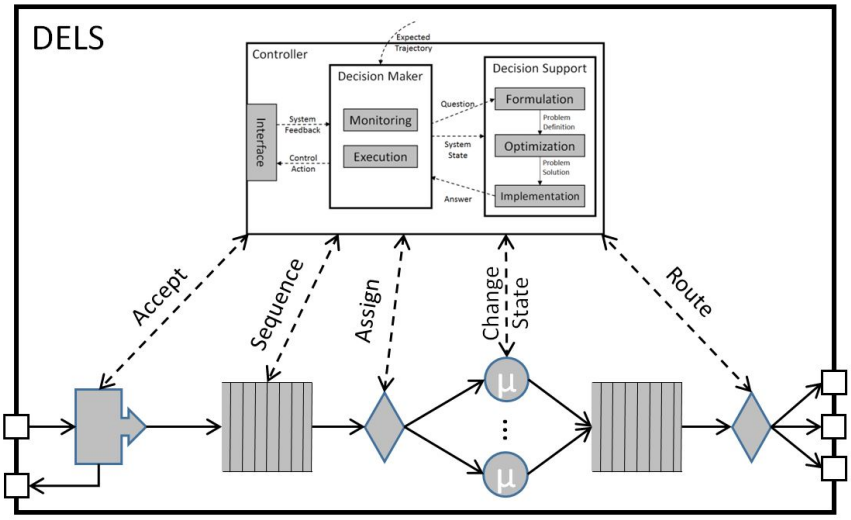


Figure 55: DELS Process Map with Process Nodes and Control 'Actuators'

The control decision and the actuator are modeled in a stylized process map of the DELS (figure 55). The component diagram in figure 56 provides an implementation pattern for the process map in figure 55, but in practice it's likely that the admissionGate is implemented with a more concrete "actuator" component that implements the required behavior; e.g. to implement the admitTask() behavior, a robotic arm is tasked to retrieve an arriving task from the AMHS conveyor and place it into a physical storage slot.

Throughout this section, a discrete event simulation model of a manufacturing workstation is constructed in SimEvents to implementing the control model by pairing a control implementation mechanism (the actuator) with a control decision mechanism (FSM). SimEvents provides a library of very primitive simulation components, which allow explicitly modeling the separation of plant and control and the actuator mechanisms in the system. Also, many of the blocks provide extension points to provide custom behavior to the block in the form of an external signal. The control behavior and decision making is implemented using the state machines and truth tables provided in the StateFlow library.

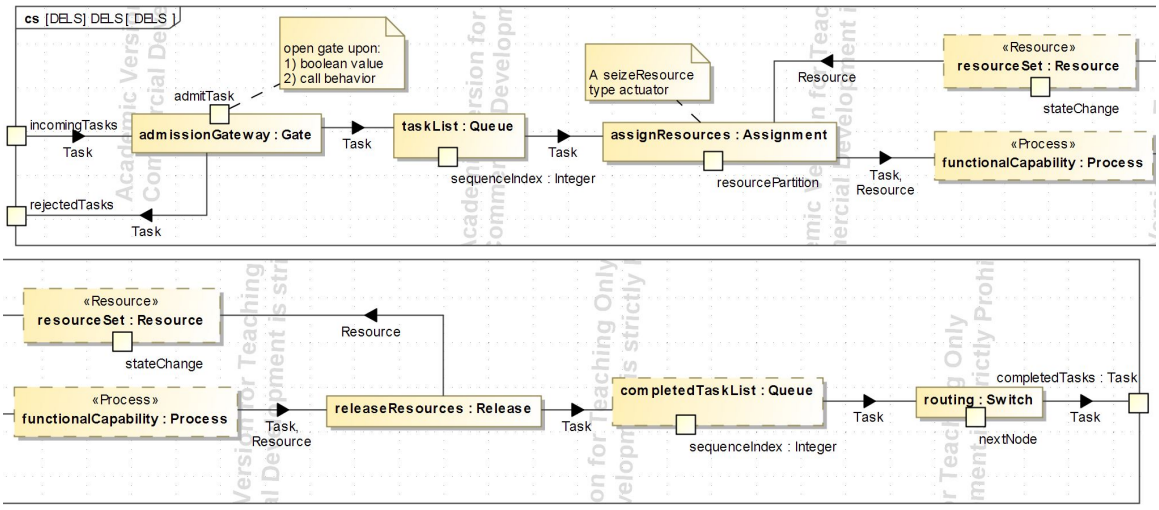


Figure 56: DELS Process Map with Process Nodes and Control 'Actuators'

Each of the following subsections focuses on implementing one of the core control questions. consisting of a computational rule (which provides the abstract specification of the execution behavior), an actuator that implements the execution behavior, and a callBehavior that the controller uses to invoke the actuator's behavior;

4.5.1 Admission Control: Which tasks get serviced?

The primary function of the admission control mechanism is to add tasks that have been admitted into the system into the system's TaskSet.

$$\text{admitTask(task)} =_{Def} \text{System.TaskSet} \cup \{task\}$$

In some system architectures, such as those that rely on token-based authentication, the

admission control decision may give the task an authorization token and the *admitTask()* behavior merely needs to check that the incoming task has this token. This mechanism provides a two-way verification process where the DELS knows that the task should be there and the task can prove that it's suppose to be admitted upon arrival.

It also provides a mechanism to dispose of or reject rejected tasks, which may be accomplished by returning them to the sender, blocking them at their origin, or storing them while the local controller queries the task's owner for further instructions.

Admission Gate In the simulation use case, the admission policy is implemented in a truth table which evaluates the state of the (local) system and the attributes of the task seeking admittance. In this model (figure 57), the admission policy only makes decisions based on the task's class, the task type (the process being requested), and the total number of tasks in the system. The threshold policy states that if the number of tasks waiting in the queue is greater than $n = 10$, then start rejecting tasks from customer class 2 (cf. section 4.3.1, figure 42). The control can be executed in a couple of different ways: as a gate implementing a yes/no boolean control decision, or as a routing block that can either route the entity into the system or out of the system.

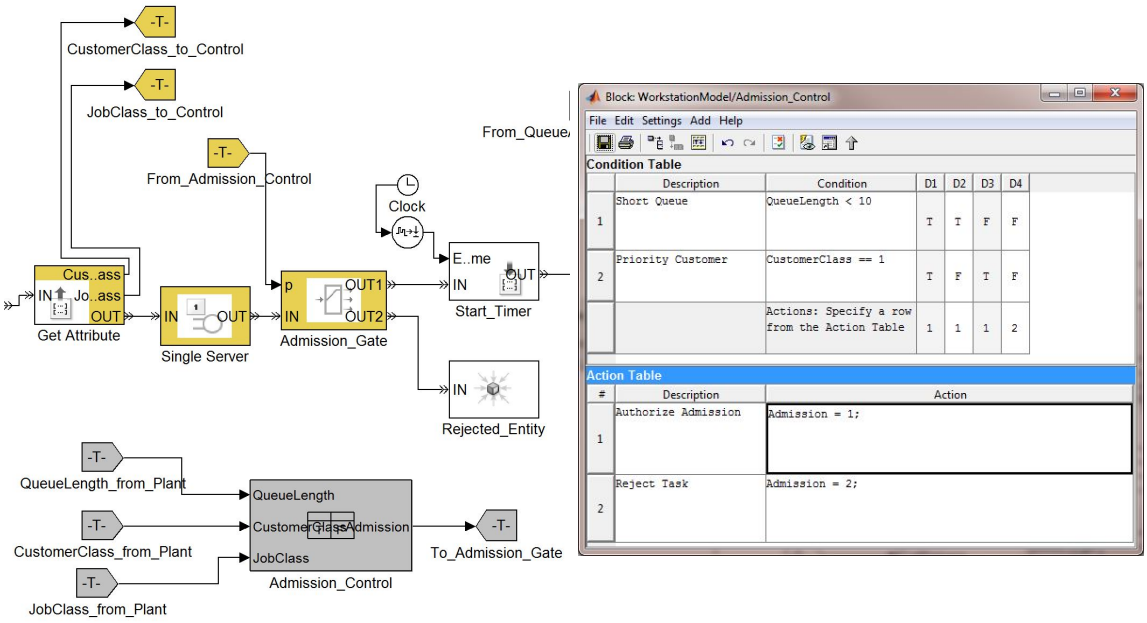


Figure 57: The admission control mechanism is implemented as a admission routing block and a truth table

4.5.2 Sequencing: When does a task get serviced?

The sequencing control decision is often thought of as a specialization of a sort function, with a sortation key defined by an attribute of the tasks, a combination of attributes, or more complex logic that may take into account the state of the System.

If $A = Index(TaskSet)$ is defined as the index of arrival for each task in the system's taskset, $\mathcal{L}_t^+$ at time t, and define $\sigma : A \to S$ as a sequencing algorithm, where $\forall k \in A$, $\sigma(k)$ is denoted a_k and that $\forall \ell_i \in \mathcal{L}_t^+$, $\ell_{a_k} \leq \ell_{a_k+1}$ where the operator $\leq$ has some ordinal or temporal definition relative to the set $TaskSet$. Then:

$$\text{sequence(TaskSet)} =_{Def} \sigma(TaskSet) = TaskSet', \text{ where } Index(TaskSet') = S$$

Therefore, whenever the system is prepared to service the next task, it only needs to select the task highest indexed task in the queue (figure 58). This execution logic implies an execution mechanism, a controllable or intelligent queue, that can find and return an arbitrary task as necessary. While some algorithms may determine only the next task to be serviced, the logic can be applied recursively to produce an implicit ordering by selecting the next first task given that the current first task isn't there.

$$[Y, S] = sort([\text{TaskSet.CapacityReq}]);$$
$$\text{TaskSet}(S(1))$$

Figure 58: The index S specifies the sequence of tasks in the TaskSet. $S(1)$ returns the index of the task that is sequenced first.

Sequencing Controlled Queues One of the major limitations of many COTS discrete event simulation tools is the limited options for queuing discipline. While there are usually workarounds available, there are some behaviors that should be inherently supported, including dynamically switching the queuing policy during the simulation run to respond to changing conditions, providing multiple layers of sequencing, like sort first by customer class then by shortest processing time (SPT); and a transparent queue to review the contents of the queue and select an arbitrary task for processing.

> A numerical priority measure is not, in general, sufficient to allow potential contractors to rank announced tasks. It assumes first, that all nodes agree on what constitutes an important task, and second, that the importance of a task can be captured in a one-dimensional quantity. [262]

In this simulation use case, several methods are demonstrated to provide insight into the desirable capabilities described above (figure fig: SimEventsQueuing). First, incoming tasks are separated by their customer class and job class to provide additional visibility into the contents of the aggregate queue. This separation process is controlled by a Queue Assignment control function and actuator that assigns task to a particular storage resource, which can segregate the tasks into any number of queues based on any arbitrary policy or set of criteria.

Second, a new object called a ControlledQueue has been created to respond to requests for information and control messages to release a particular task. The ControlledQueue is capable of being queried for information relevant to the decision algorithm. In this example, the CallReview function returns the waiting time and process time of the entity that has been in the queue the longest. There is a control mechanism ReviewReleaseControl that reviews the availability of the servers and when a server becomes available, send a

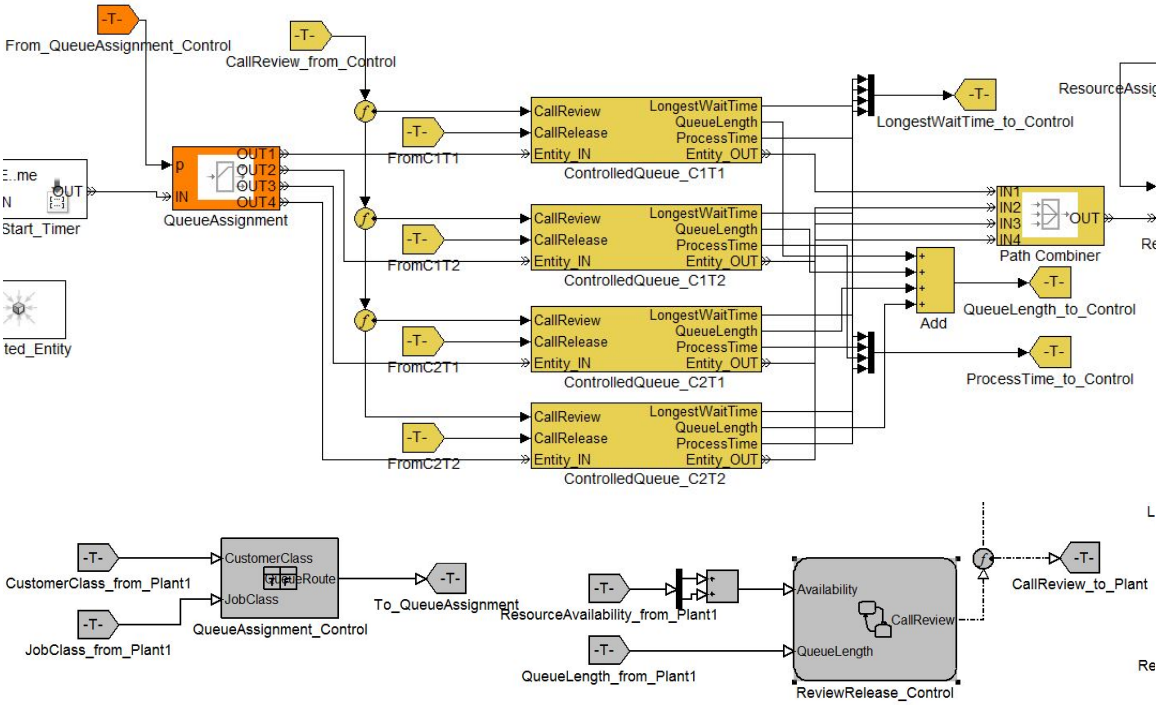


Figure 59: Implementing more complex sequencing rules in SimEvents requires assigning tasks to separate queues based on their type and implementing a review and release mechanism for each queue.

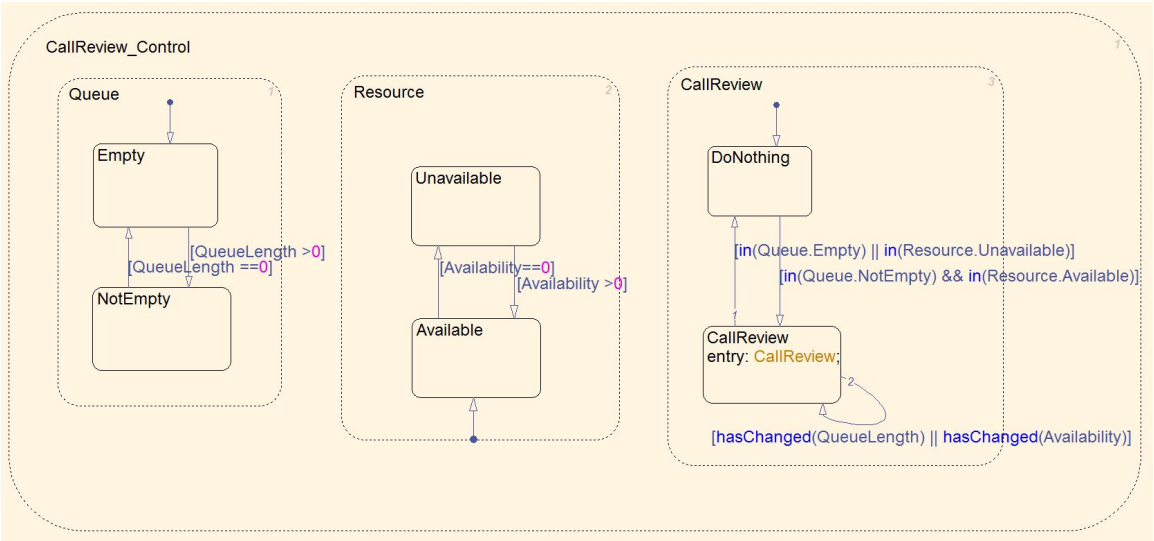


Figure 60: The review and release mechanism for the collection of controlled queues is implemented as a state machine.

CallReview message to each queue as well as the Scheduler. The scheduler then will evaluate its control algorithm (or policy) and decide which class of task should be released. It then

and sends a CallRelease request to the queue that contains the desired task.

There are some obvious shortcomings to this facet of the simulation, including that only the first (oldest) entity in each queue is reviewed and the entities need to be separated into many queues to facilitate reviewing a large enough selection of the entities to make an adequate decision, which suggest a cumbersome implementation that will not scale well as the number of classes and types of tasks increases. However, a transparent queue with an arbitrary sequencing mechanism has demonstrated benefits and remains the goal for executing control decisions in simulation tools.

4.5.3 Assignment: Which resource services the task?

Assigning a task to be serviced by a particular resource(s) places that task in the resource's taskSet, either logically or physically. It should be clear that tasks can be sequenced and assigned to the a particular resource without ever physically moving from their queueing storage slot.

$$\text{assign(resource, task)} =_{Def} \text{Resource.TaskSet} \cup \{task\}$$

In some cases where the task is only logically assigned to the resource, when the resource is ready to process the task, the task may be transferred to the resource, i.e. placed in the resource's local queue, or the resource may move to the location that the task occupies.

Resource Assignment & Scheduling In this example, the controller uses a scheduling strategy that implements a generalized $c\mu$ rule [287]. Therefore, the controller needs to know the expected processing time of each task, which are different for each class of tasks, and how long each task has been waiting. Moreover, the controller must decide which fixed resource will be assigned to process the task (figure fig: SimEventsResourceAssigment). Due to different processing rates of each fixed resource, the set-up state of the resource or the processing capabilities of each resource; it may be optimal to delay the processing of the highest priority job until a particular resource becomes available.

Ideally, all the entities would be in a single, transparent queue that can be queried for the details on every task in the queue. Then the controller produces a sequence and resource

assignment for all of the tasks. However, in this demonstration the controller determines the sequence to process the oldest task from each of the four queues and assignment to a fixed resource. When the assigned resource becomes available, the controller calls the queue to release that task and routes it to the desired resource.

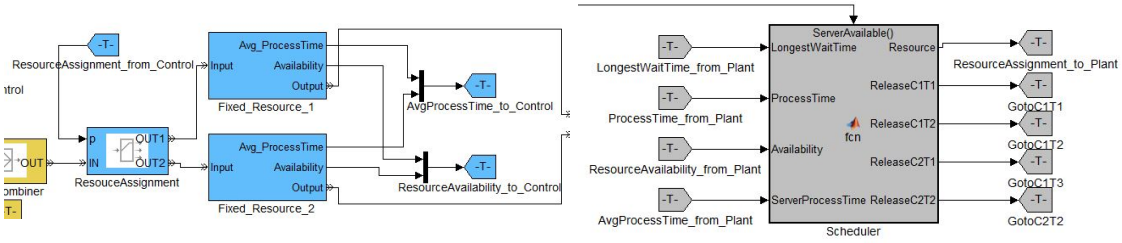


Figure 61: The scheduler (right) determines both the next task to be serviced and which resource it is assigned to (left)

4.5.4 Routing: Where does a task go after service?

The routing control decision is focused on selecting a process plan or modifying an existing process plan; therefore, the result of this decision process is that the callBehavior is given some information about the nextProcess or the ability to evaluate next(processPlan').

$$processPlan' = f(processPlan)$$

$$nextProcess = next(processPlan')$$

$$targetFlowNode = g(nextProcess)$$

From the DELS' perspective, the outcome of this decision is either deciding which output flow node or output port to place the task on, or contracting a MHS to move the task to its destination. From the task's perspective, it's a selection of a DELS to perform the next(process) in its process plan (which may be part of this question unto itself if there are multiple AND options). The key distinction between this control decision and the resource assignement decision is that in the assignment decision the resource is a part of my local resourceSet, but in the routing case, the controller is negotiating with an autonomous peer controller. Routing in the case of generalized process plan selection is very difficult, but it is the fundamental mechanism underlying distributed decision making.

In the control strategy sense, the process plan can be selected myopically, e.g. least cost processor capable of executing the next processing task, or in a more global manner that may select the expected shortest path through the complete, or some subset, of the process plan. It may require additional negotiation and solicitation modules to facilitate true distributed capabilities. However, the DELS may only need to decide the DELS or resource to which it must route the task. Then it is the responsibility of the MHS offering the move service to decide how, i.e. which flow edge and which transport resource, to move the task from its currentLocation to the required destination, see e.g. [168].

Outbound Routing The routing control mechanism reviews the process plan to determine the next process node which to send the task (figure 62). For this control decision, the controller may have several options available to it based on the plan (or task graph), where it may be required to determine, based on the information available to it, the shortest path through the remaining subtasks that the job must be routed.

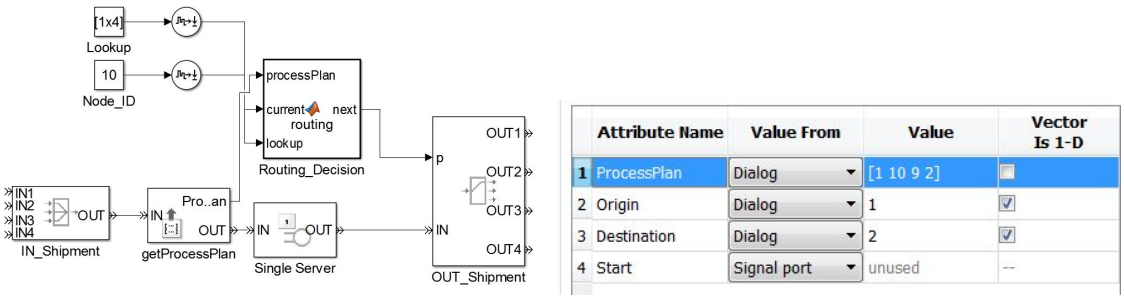


Figure 62: The routing decision examines the process plan stored in the task

4.5.5 Change The State of Your Resource: Set-up/Tear-down/ Maintenance

Since discrete-state resources are commonly modeled as finite state machines in SEMI, GEM, or CAMX, the change state mechanism closely mimics the finite state machine transition function $T : S \times \Sigma \rightarrow S$, where given a state of the system S and input Σ then the system transitions to the next state S. The callBehavior *changeState()* prompts the resource to transition to $S =$ newState. In many cases however, this is transition is executed asynchronously (contrary to FSM execution behavior) due to the time and additional auxiliary resources required to set-up, maintain, or re-position the resource. Moreover, there's

not really an actuator required here since the resource can induce its own state change.

$$\text{Resource.}\textit{changeState}(\text{newState}) \implies \text{Resource.CurrentState} = \text{newState}$$

Change The State of Your Resource The controller also implements a maintenance policy, which responds to preemptive failures (figure 63). It is implemented as a finite state machine that generates overhead maintenance and set-up tasks, and signals the resource's availability to the system. The current implementation allows for flexible extension to add additional sources of failures and layers of maintenance.

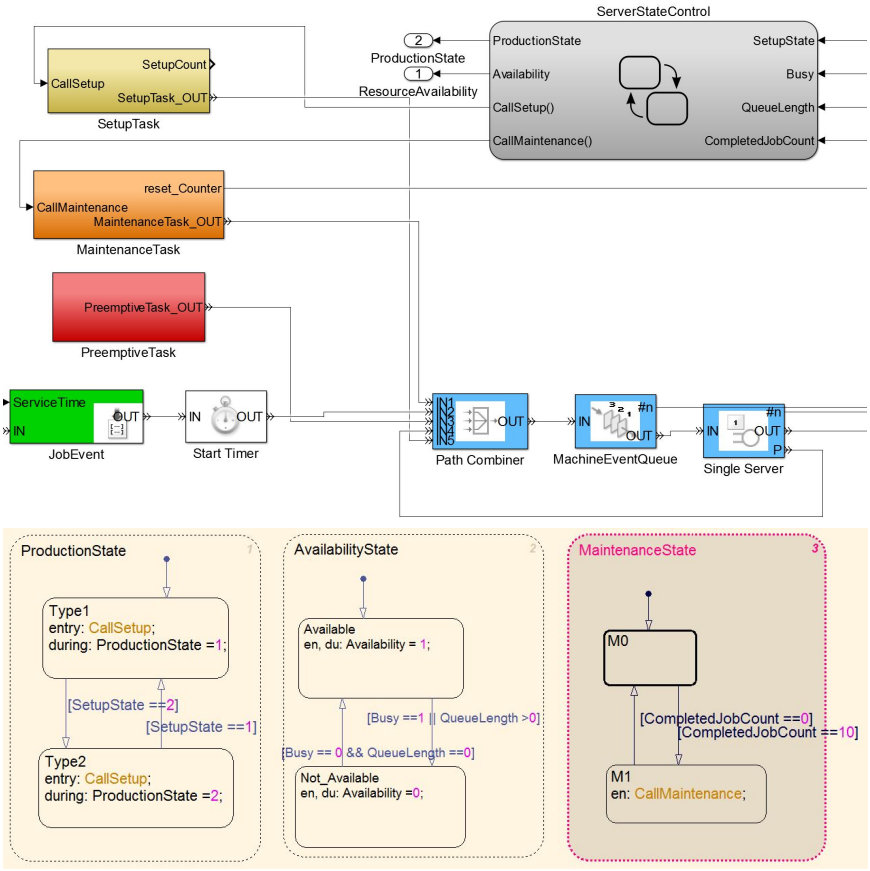


Figure 63: The change state mechanism in SimEvents is implemented using a state machine (bottom) that generates overhead tasks for the resource to process (top)

4.6 Summary

To answer the control questions posed in chapter 3, a round-trip analysis methodology is constructed in this chapter, which provides the functional specification of the DELS

controller (figure 64). Whereas the control questions define a comprehensive specification of all the decision-making mechanisms that the controller should be able to provide, the round-trip analysis methodology defines a comprehensive functional specification of all the methods that the controller needs to be able to make those decisions.

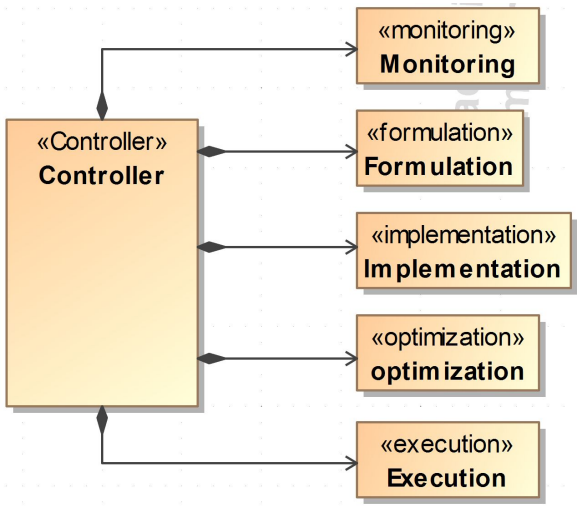


Figure 64: The functional architecture of the controller is defined be the set of of functional components that implement a roundtrip analysis methdology for each control question.

This methodology narrows the gap between operations research methods and practical applications by establishing interoperability not only between system models and analysis models, but also between analysis models and execution systems. Furthermore, the functional components constructed in this chapter can be re-organized into a question-driven functional decomposition of the system, which can be incorporated into a design methodology for DELS controllers.

Future work should focus on defining the uniform and robust interfaces between the functional components defined in this chapter. Well-defined interfaces enable each of these components to be implemented individually and therefore support a variety of architectures that integrate human, human-in-the-loop, and automated or intelligent decision-making methods; e.g. an automated monitoring module can be integrated with a human who formulates and solves ad-hoc analyses to respond to excursions, and vice versa, a human monitoring the system that can access a decision support module.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1 Contributions

While addressing the need for a model of ISA-95/L3 control, it became clear that constructing a controller architecture that is extensible and reusable across the DELS domain required addressing more fundamental gaps in the DELS and operational control literature. Ultimately the controller architecture itself can be viewed as an application or usage of the research developed in this dissertation.

The fundamental contribution of this dissertation is concerned with interoperability and bridging the gap between operations research analysis models and practical applications of the results. This dissertation closes this gap by constructing a standard domain-specific language, standard control problem definitions, and a standard analysis methodology.

The domain specific language developed in chapter 2 meets a broader requirement for facilitating interoperability for DELS, including system integration, plug-and-play analysis methods and tools, and system design methodologies. The domain-specific language formalizes a recurring product, process, resource, and facility description of the DELS domain. It provides a common language to discuss our systems, including the questions that we want to ask about our systems, the problems that we need to solve in order to answer those questions, and the mechanisms to deploy the solution.

While the DSL fulfills a more general interoperability objective, the control questions presented in chapter 3 refine and extend the interoperability mechanism between system and analysis models by formulating a canonical functional description of operational control. This fundamental set of control questions is formalized by a standard abstraction formulated using the abstract PPRF semantics. The questions themselves form a comprehensive functional specification of all the decision-making mechanisms that a controller needs to be able to provide; i.e. a model of analysis models or a metamodel of operational control.

A standard representation of each class of control problems is only a partial solution to fully addressing operational control. The final contribution of this dissertation (chapter 4) constructs a round-trip analysis methodology that completes the bridge between operations research analysis models and deployable control mechanisms. This contribution not only formalizes a functional analysis pathway from formulating an analysis model to executing a control actions, but also integrates well-tested mechanisms for implementing each of the functional components. This round-trip analysis methodology is grounded in a more fundamental insight into how analysis is executed for operational control decision-making and prescribes a functional architecture for the DELS controller.

One result of these contributions is a pathway to a uniform controller architecture for operational control of DELS; this allows for the integration of DELS domains and the development of a flexible and extensible controller architecture that can extend from centralized, hierarchical control to distributed, agent-based control. The controller architecture provides the functional scaffolding that supports the more fundamental concept of round-trip analysis modeling from formulation to execution.

5.2 Scope and Limitations

- *Scope of the domain-specific language:* As stated upfront in the title, the domain-specific language and control questions are tailored to the domain of DELS. While there may be systems that have not been discussed in this dissertation that can be described as DELS, the more complicated use cases are the exceptions that arise when a control problem or system definition from the DELS domain does not fit neatly into the modeling framework constructed in this dissertation. In these cases, should the DELS modeling language be extended to accommodate the unique features of the system definition or should they be pigeon-holed into the existing language? This is a modeling decision that needs careful consideration because it focuses on which systems can and should be defined as DELS while maintaining the broad applicability and stability of the core modeling language. Additionally, is there a boundary (conceptual or otherwise) where the DELS language must be specialized to accommodate specific

features of each sub-class of DELS, e.g. manufacturing specific concepts that do not apply more broadly.

- *Role of artificial intelligence techniques:* The modeling methodologies discussed and developed throughout this dissertation are focused on bridging the gap between *operations research* theory and application, i.e. structured decision making. Artificial intelligence, learning methods, and other techniques from computer science that do not require a rigorous decision support framework may offer a complementary approach to operations research methodologies, but are outside of the scope of the analysis methodology discussed in chapter 4.

- *Modeling choices and solution methods:* The control questions and formulation component of the controller assume that if a class of problems can be formulated uniformly then each instance of the control problem, regardless of the original domain, can be solved using a common set of algorithms and tools. While this is the approach taken in the development of operations research methods, certainly it is possible that the way the system is modeled and abstracted has an impact on the quality or feasibility of the solution, i.e. by abstracting key details of the problem, the result may be a solution that is not feasible or implementable in the original target system.

- *Axiomatic definitions of behavior and control:* This dissertation was careful to imply that the domain-specific language and control questions formed a canonical description of the domain, rather than constructing an axiomatic definition of the domain. While many operations research techniques are rooted in precise mathematical models and methods, domain-specific applications have not been defined axiomatically. This does not imply that the domain-specific semantics cannot be defined axiomatically, and in fact, many of the well-formedness rules captured in the DSL offer a few thoughts on how to approach this problem. This dissertation definitely draws a line between canonical and axiomatic definitions of behavior and control.

- *Bridging the gap between theory and application:* Consistent with the theme of this

dissertation, there is a question about bridging the gap between the abstract methods discussed in this dissertation and concrete applications; i.e., how is this research disseminated and applied to bridge the gap between operations research and tool implementations? The conceptual elements of the language can be taught to students at all levels and applied by practitioners to connect methods and tools. The abstract diagrams discussed in chapter 2 would be maintained by experts (or a standards body), who apply the language to construct modeling tools and libraries which are accessible to users to construct system and analysis models; e.g. a discrete event simulation tool implements the DELS DSL as model libraries which are accessible to users of the tool.

In addition to developing methods and tools using the DSL, there is an open question of how to make the extant research accessible and, when such cases arise, who interprets the intent of semantically incomplete system and analysis models in the literature.

5.3 Future Work

Possibilities for future work include:

- *Enhancing the DELS domain-specific language:* The DELS DSL is constructed by abstracting the modeling elements that are common across the DELS domain, and as such, there remains additional work to refine and enhance the language itself. While the product, process, resource, and facility pattern represents a canonical abstraction of the core concepts for this domain, the properties of these metaclass definitions must reflect the broad range of systems in the DELS domain. The language must accomplish the following: (1) extend operations research analysis modeling beyond bill-of-material definitions and bridge the gap to product definition interoperability standards, (2) select a robust and expressive language for specifying process plans, and (3) incorporate greater detail for defining facility layout semantics from standards such as CMSD. There is a significant amount of work to incorporate existing system and analysis description languages while maintaining a language that is broadly applicable to DELS.

- *Implementations of the controller:* Optimal-control design methods and tools in other engineering domains provide a direct pathway from the design and analysis of controllers to prototyping and testing in a controlled environment and finally to deploying the code in the system being designed. Many current generation DELS control applications, such as MES and WCS, are already software-driven and would benefit from a design methodology and a set of corresponding tools to develop, test, and deploy control methods and tools.

  As an abstract functional architecture, the controller functional architecture developed in chapter 4 focuses on the round-trip analysis methodology and many of the components required to implement the methodology, and therefore developing a working prototype of the controller along with a test bed of system models and scenarios would provide a sandbox to experiment with control methods and tools. Developing this prototype would also require the development of novel supporting analysis methods and tools, e.g. the controller must be able to interface with a set of base system simulation models that simulate expected feedback from the system.

- *Exploration of structure, behavior, and control:* The modeling language for DELS is constructed in three basic layers: structure is provided by TFN, behavior is provided by PPRF which is extended from or layered on top of the TFN, and finally the control questions and decision-making architecture is top most layer. Further investigation needs to be focused on exploring the fundamental nature of this multi-layer architecture as a canonical layered abstraction for both analysis models and design methodologies: e.g, (1) Does this multi-layer architecture provide a broad hierarchy for designing and analyzing DELS?, (2) Are there fundamental decision or design problems that can only be answered at each layer of abstraction?, and (3) How can the layers of abstraction be applied to organize multi-fidelity analysis models?

- *Foundation for a discrete event simulation language:* Does this language provide a complete and more natural language for discrete event simulation (DES) for DELS? In disciplines such as mechanical and aerospace engineering, there is a clear separation

of the plant from the control mechanism throughout the design process. Accordingly, tools such as Modelica and Simulink provide a modeling and analysis environment that supports specifying the plant and control separately. In fact, Simulink supports the generation of control code from its simulation environment, which then can be implemented in real-life systems.

The modeling and simulation environment and the supporting tools are significantly different in the DELS domain. In many existing commercial off the shelf (COTS) DES tools that support the modeling of DELS, the plant and control are conflated and capabilities for explicit modeling of control are limited. Tool support for explicit separation of plant and control has the potential to instigate new avenues of modeling and analysis and make existing research more widely accessible and implementable. These ideas include in-the-loop simulation coupled to a real system or integrate optimization tools directly into the simulation environment.

Also, often implementing a conceptual model in a particular discrete event simulation tool requires compromises, or work-arounds, to capture the desired system behavior given the limitations of the language or provided modeling constructs. Instead the language should natively support the systems as designed, allowing the code that will be used to control the actual system to be designed and debugged it in a high-fidelity test environment. The metamodel of control developed in this dissertation has the potential to change the way that analysis tools are structured, discrete event simulation in particular.

References

[1] Reference Architecture Foundation for Service Oriented Architecture Version 1.0, December 2012. URL `http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.html`.

[2] Ajith Abraham. Rule-based expert systems. *Handbook of measuring system design*, 2005.

[3] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[4] Donald L Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6(1):40–54, 1977.

[5] Rakesh Agrawal, Roberta J Cochrane, and Bruce G Lindsay. On maintaining priorities in a production rule system. Technical report, Systems Research Center, University of Maryland, College Park, 1991.

[6] Vipul Agrawal, Xiuli Chao, and Sridhar Seshadri. Dynamic balancing of inventory in supply chains. *European Journal of Operational Research*, 159(2):296–317, 2004.

[7] Can Akkan. Finite-capacity scheduling-based planning for revenue-based capacity management. *European Journal of Operational Research*, 100(1):170–179, 1997.

[8] Gad Allon and Awi Federgruen. Competition in service industries. *Operations Research*, 55(1):37–55, 2007.

[9] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.

[10] Enrico Angelelli and Maria Grazia Speranza. The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, 137(2):233–247, 2002.

[11] Aruna Apte. Humanitarian logistics: A new field of research and action. information and operations managment, 3 (1), 1-100, 2009.

[12] Chid Apte. The role of machine learning in business optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1–2, 2010.

[13] Kenneth J Arrow, Theodore Harris, and Jacob Marschak. Optimal inventory policy. *Econometrica: Journal of the Econometric Society*, pages 250–272, 1951.

[14] Anteneh Ayanso, Moustapha Diaby, and Suresh K Nair. Inventory rationing via drop-shipping in internet retailing: A sensitivity analysis. *European Journal of Operational Research*, 171(1):135–152, 2006.

[15] Haldun Aytug, Mark A Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110, 2005.

[16] M Naceur Azaiez and SS Al Sharif. A 0-1 goal programming model for nurse scheduling. *Computers & Operations Research*, 32(3):491–507, 2005.

[17] James Bailey, Alexandra Poulovassilis, and Peter T Wood. Analysis and optimisation of event-condition-action rules on xml. *Computer Networks*, 39(3):239–259, 2002.

[18] Nagraj Balakrishnan, V Sridharan, and J Wayne Patterson. Rationing capacity between two product classes. *Decision Sciences*, 27(2):185–214, 1996.

[19] Egon Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations research*, 17(6):941–957, 1969.

[20] Burcu Balcik, Benita M Beamon, and Karen Smilowitz. Last mile distribution in humanitarian relief. *Journal of Intelligent Transportation Systems*, 12(2):51–63, 2008.

[21] Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(6):1205–1220, 2012.

[22] Gülay Barbarosoğlu, Linet Özdamar, and Ahmet Cevik. An interactive approach for hierarchical analysis of helicopter logistics in disaster relief operations. *European Journal of Operational Research*, 140(1):118–133, 2002.

[23] Jay Barney. Firm resources and sustained competitive advantage. *Journal of management*, 17(1):99–120, 1991.

[24] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based runtime verification. In *Verification, Model Checking, and Abstract Interpretation*, pages 44–57. Springer, 2004.

[25] JoseM Bastos. Batching and routing: two functions in the operational planning of flexible manufacturing systems. *European Journal of Operational Research*, 33(3): 230–244, 1988.

[26] Carrie Beam and Arie Segev. Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik*, 39(3):263–268, 1997.

[27] Benita M. Beamon. Supply chain design and analysis: Models and methods. *International Journal of Production Economics*, 55(3):281–294, 1998. URL `http://www.sciencedirect.com/science/article/pii/S0925527398000796`.

[28] Jeroen Beliën and Erik Demeulemeester. Building cyclic master surgery schedules with leveled resulting bed occupancy. *European Journal of Operational Research*, 176 (2):1185–1204, 2007.

[29] JWM Bertrand and V Sridharan. A study of simple rules for subcontracting in make-to-order manufacturing. *European Journal of Operational Research*, 128(3):509–531, 2001.

[30] Christian Bierwirth and Frank Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, 2010.

[31] John H Blackstone, Don T Phillips, and Gary L Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *The International Journal of Production Research*, 20(1):27–45, 1982.

[32] Basilio Bona, Paolo Brandimarte, Cosimo Greco, and Giuseppe Menga. Hybrid hierarchical scheduling and control systems in manufacturing. *IEEE Transactions on Robotics and Automation*, 6(6):673–686, 1990.

[33] Edward H Bowman. The schedule-sequencing problem. *Operations Research*, 7(5): 621–624, 1959.

[34] Nils Boysen, Malte Fliedner, and Armin Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2):349–373, 2009.

[35] Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3):157–183, 1993.

[36] Paolo Brandimarte. Exploiting process plan flexibility in production scheduling: A multi-objective approach. *European Journal of Operational Research*, 114(1):59–71, 1999.

[37] Gerald G Brown and Antonios L Vassiliou. Optimizing disaster relief: real-time operational and tactical decision support. Technical report, DTIC Document, 1993.

[38] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[39] Robert L Burdett and Erhan Kozan. A disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research*, 200(1): 85–98, 2010.

[40] Kathryn E Caggiano, John A Muckstadt, and James A Rappold. Integrated real-time

capacity and inventory allocation for reparable service parts in a two-echelon supply system. *Manufacturing & Service Operations Management*, 8(3):292–319, 2006.

[41] Ann Campbell, Lloyd Clarke, Anton Kleywegt, and Martin Savelsbergh. The inventory routing problem. In *Fleet management and logistics*, pages 95–113. Springer, 1998.

[42] Yves Caseau and François Laburthe. Improved clp scheduling with task intervals. In *ICLP*, pages 369–383. Citeseer, 1994.

[43] C Richard Cassady and Erhan Kutanoglu. Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Transactions on Reliability*, 54(2):304–309, 2005.

[44] Christos G. Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer, 2008.

[45] Bryan A Catron and Steven R Ray. Alps: A language for process specification. *International Journal of Computer Integrated Manufacturing*, 4(2):105–113, 1991.

[46] Kyle D Cattani and Gilvan C Souza. Inventory rationing and shipment flexibility alternatives for direct market firms. *Production and Operations Management*, 11(4): 441–457, 2002.

[47] Aakil M Caunhye, Xiaofeng Nie, and Shaligram Pokharel. Optimization models in emergency logistics: A literature review. *Socio-Economic Planning Sciences*, 46(1): 4–13, 2012.

[48] Sila Çetinkaya and Chung-Yee Lee. Stock replenishment and shipment scheduling for vendor-managed inventory systems. *Management Science*, 46(2):217–232, 2000.

[49] Pankaj Chandra and Marshall L Fisher. Coordination of production and distribution planning. *European Journal of Operational Research*, 72(3):503–517, 1994.

[50] Fangruo Chen. Market segmentation, advanced demand information, and supply chain performance. *Manufacturing & Service Operations Management*, 3(1):53–67, 2001.

[51] Zhi-Long Chen. Integrated production and outbound distribution scheduling: review and extensions. *Operations Research*, 58(1):130–148, 2010.

[52] Jinxing Cheng, Michael Gruninger, Ram D Sriram, and Kincho H Law. Process specification language for project scheduling information exchange. *International Journal of IT in Architecture Engineering and Construction*, 1:307–328, 2003.

[53] Junzilan Cheng, John Fowler, Karl Kempf, and Scott Mason. Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. *Computers & Operations Research*, 53:275–287, 2015.

[54] Nicos Christofides. The vehicle routing problem. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 10(1):55–70, 1976.

[55] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282, 1981.

[56] Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management science*, 6(4):475–490, 1960.

[57] G. Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[58] Edward G Coffman, Jr, Michael R Garey, and David S Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[59] Christian Colombo, Gordon J Pace, and Gerardo Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *Formal Methods for Industrial Critical Systems*, pages 135–149. Springer, 2009.

[60] Teodor Gabriel Crainic, Michel Gendreau, and Pierre Dejax. Dynamic and stochastic

models for the allocation of empty containers. *Operations Research*, 41(1):102–126, 1993.

[61] Teodor Gabriel Crainic, Guido Perboli, Walter Rei, and Roberto Tadei. Efficient lower bounds and heuristics for the variable cost and size bin packing problem. *Computers & Operations Research*, 38(11):1474–1482, 2011.

[62] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. Springer, 2001.

[63] Stéphane Dauzère-Pérès and Jan Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306, 1997.

[64] Randall Davis, Bruce Buchanan, and Edward Shortliffe. Production rules as a representation for a knowledge-based consultation program. *Artificial intelligence*, 8(1): 15–45, 1977.

[65] Wayne Davis, Albert Jones, and Abdol Saleh. Generic architecture for intelligent control systems. *Computer Integrated Manufacturing Systems*, 5(2):105–113, 1992.

[66] Prabuddha De, Jay B Ghosh, and Charles E Wells. Job selection and sequencing on a single machine in a random environment. *European Journal of Operational Research*, 70(3):425–431, 1993.

[67] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501, 2007.

[68] Francis De Vericourt, Fikri Karaesmen, and Yves Dallery. Dynamic scheduling in a make-to-stock system: A partial characterization of optimal policies. *Operations Research*, 48(5):811–819, 2000.

[69] Brian C Dean, Michel X Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.

[70] Pierre J Dejax and Teodor Gabriel Crainic. Survey paper-a review of empty flows and fleet management models in freight transportation. *Transportation Science*, 21 (4):227–248, 1987.

[71] Jeroen Dejonckheere, Stephen M Disney, Marc R Lambrecht, and Denis R Towill. Measuring and avoiding the bullwhip effect: A control theoretic approach. *European Journal of Operational Research*, 147(3):567–590, 2003.

[72] Rommert Dekker and Moritz Fleischmann. *Reverse logistics: quantitative models for closed-loop supply chains*. Springer, 2004.

[73] Ivan M Delamer and Jose L Martinez Lastra. Service-oriented architecture for distributed publish/subscribe middleware in electronics production. *IEEE Transactions on Industrial Informatics*, 2(4):281–294, 2006.

[74] Berend Denkena, Hans Kurt Tönshoff, Michael Zwick, and Peer-Oliver Woelk. Process planning and scheduling with multiagent systems. In *Knowledge and Technology Integration in Production and Services*, pages 339–348. Springer, 2002.

[75] Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M. Solomon, and Francois Soumis. *The VRP with pickup and delivery*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[76] Vinayak Deshpande, Morris A Cohen, and Karen Donohue. A threshold inventory rationing policy for service-differentiated demand classes. *Management Science*, 49 (6):683–703, 2003.

[77] Martin Desrochers, Jan Karel Lenstra, and Martin WP Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46(3):322–332, 1990.

[78] Franklin Dexter, Alex Macario, and Rodney D Traub. Optimal sequencing of urgent surgical cases. *Journal of clinical monitoring and computing*, 15(3-4):153–162, 1999.

[79] Franklin Dexter, Richard H Epstein, Eric Marcon, and Renato de Matta. Strategies to reduce delays in admission into a postanesthesia care unit from operating rooms. *Journal of PeriAnesthesia Nursing*, 20(2):92–102, 2005.

[80] DM Dilts, NP Boyd, and HH Whorms. The evolution of control architectures for automated manufacturing systems. *Journal of Manufacturing Systems*, 10(1):79–93, 1991.

[81] Stephen Michael Disney and Denis Royston Towill. A discrete transfer function model to determine the dynamic stability of a vendor managed inventory supply chain. *International Journal of Production Research*, 40(1):179–204, 2002.

[82] Amine Drira, Henri Pierreval, and Sonia Hajri-Gabouj. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007.

[83] Moshe Dror and Pierre Trudeau. Split delivery routing. *Naval Research Logistics (NRL)*, 37(3):383–402, 1990.

[84] A Dugenske, A Fraser, T Nguyen, and R Voitus. The national electronics manufacturing initiative (nemi) plug and play factory project. *International Journal of Computer Integrated Manufacturing*, 13(3):225–244, 2000.

[85] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[86] MJR Ebben, EW Hans, and FM Olde Weghuis. Workload based order acceptance in job shop environments. *OR spectrum*, 27(1):107–122, 2005.

[87] Lisa M Ellram. The supplier selection decision in strategic partnerships. *Journal of Supply Chain Management*, 26(4):8, 1990.

[88] Wedad Elmaghraby and Pınar Keskinocak. Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions. *Management Science*, 49(10):1287–1309, 2003.

[89] Hoda A ElMaraghy. Automated tool management in flexible manufacturing. *Journal of Manufacturing Systems*, 4(1):1–13, 1985.

[90] Fadi George Fadel, Mark S Fox, and Michael Gruninger. A generic enterprise resource ontology. In *Proceedings of Third Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 117–128. IEEE, 1994.

[91] Peyman Faratin, Carles Sierra, and Nick R Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3):159–182, 1998.

[92] Awi Federgruen and Paul Zipkin. A combined vehicle routing and inventory allocation problem. *Operations Research*, 32(5):1019–1037, 1984.

[93] Awi Federgruen, H Groenevelt, and Hendrik Cornelis Tijms. Coordinated replenishments in a multi-item inventory system with compound poisson demands. *Management Science*, 30(3):344–357, 1984.

[94] Steven J Fenves, Sebti Foufou, Conrad Bock, and Ram D Sriram. Cpm2: a core model for product data. *Journal of Computing and Information Science in Engineering*, 8 (1):014501, 2008.

[95] Frank Fiedrich, Fritz Gehbauer, and U Rickers. Optimized resource allocation for emergency response after earthquake disasters. *Safety Science*, 35(1):41–57, 2000.

[96] Gene Fliedner. Cpfr: an emerging supply chain tool. *Industrial Management & Data Systems*, 103(1):14–21, 2003.

[97] Mark S Fox, Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modelling: preliminary concepts for linking structure and behaviour. In *Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 71–81. IEEE, 1995.

[98] Guillermo Gallego and Garrett Van Ryzin. A multiproduct dynamic pricing problem and its applications to network yield management. *Operations Research*, 45(1):24–41, 1997.

[99] Brendan Galloway and Gerhard P Hancke. Introduction to industrial control networks. *Communications Surveys & Tutorials, IEEE*, 15(2):860–880, 2013.

[100] Charles Galunic and Simon Rodan. *Resource recombinations in the firm: knowledge structures and the potential for Schumpeterian innovation.* INSEAD, 1997.

[101] Mansour Abou Gamila and Saeid Motavalli. A modeling technique for loading and scheduling problems in fms. *Robotics and Computer-Integrated Manufacturing*, 19(1): 45–54, 2003.

[102] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Pearson Education, 1994.

[103] Michael R Garey, Ronald L Graham, David S Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[104] Arthur M Geoffrion and Glenn W Graves. Multicommodity distribution system design by benders decomposition. *Management science*, 20(5):822–844, 1974.

[105] A Gharbi and JP Kenne. Production and preventive maintenance rates control for a manufacturing system: an experimental design approach. *International Journal of Production Economics*, 65(3):275–287, 2000.

[106] Soumen Ghosh, Steven A Melnyk, and Gary L Ragatz. Tooling constraints and shop floor scheduling: evaluating the impact of sequence dependency. *The International Journal of Production REsearch*, 30(6):1237–1253, 1992.

[107] Donald W Gillies and Jane W-S Liu. Scheduling tasks with and/or precedence constraints. *SIAM Journal on Computing*, 24(4):797–810, 1995.

[108] Gregory A Godfrey and Warren B Powell. An adaptive dynamic programming algorithm for dynamic fleet management, i: Single period travel times. *Transportation Science*, 36(1):21–39, 2002.

[109] Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

[110] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[111] Robert M Grant. The resource-based theory of competitive advantage: implications for strategy formulation. *Knowledge and Strategy*, 33(3):3–23, 1991.

[112] Ann E Gray, Abraham Seidmann, and Kathryn E Stecke. A synthesis of decision models for tool management in automated manufacturing. *Management Science*, 39 (5):549–567, 1993.

[113] Donald Gross, Douglas R Miller, and Richard M Soland. A closed queueing network model for multi-echelon repairable item provisioning. *IIE Transactions*, 15(4):344–352, 1983.

[114] Tonci Grubic and Ip-Shing Fan. Supply chain ontology: Review, analysis and synthesis. *Computers in Industry*, 61(8):776–786, 2010.

[115] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177 (1):1–21, 2007.

[116] Jinxiang Gu, Marc Goetschalckx, and Leon F McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203(3):539–549, 2010.

[117] P Gu, S Balasubramanian, and DH Norrie. Bidding-based process planning and

scheduling in a multi-agent system. *Computers & Industrial Engineering*, 32(2):477–496, 1997.

[118] V Daniel R Guide Jr. Production planning and control for remanufacturing: industry practice and research needs. *Journal of Operations Management*, 18(4):467–483, 2000.

[119] Tzeng Gwo-Hshiung. Multiple attribute decision making: methods and applications. *Multiple Attribute Decision Making: Methods and Applications*, 2010.

[120] Albert Y Ha. Inventory rationing in a make-to-stock production system with several demand classes and lost sales. *Management Science*, 43(8):1093–1103, 1997.

[121] Albert Y Ha. Optimal dynamic scheduling policy for a make-to-stock production system. *Operations Research*, 45(1):42–53, 1997.

[122] Steven T Hackman and Robert C Leachman. A general framework for modeling production. *Management Science*, 35(4):478–495, 1989.

[123] Ali Haghani and Sei-Chang Oh. Formulation and solution of a multi-commodity, multi-modal network flow model for disaster relief operations. *Transportation Research Part A: Policy and Practice*, 30(3):231–250, 1996.

[124] Mark Haimovich and AHG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

[125] Milton Harris and Artur Raviv. A theory of monopoly pricing schemes with demand uncertainty. *The American Economic Review*, pages 347–365, 1981.

[126] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[127] Warren H Hausman and Gary D Scudder. Priority scheduling rules for repairable inventory systems. *Management Science*, 28(11):1215–1232, 1982.

[128] Warren H Hausman, Leroy B Schwarz, and Stephen C Graves. Optimal storage assignment in automatic warehousing systems. *Management Science*, 22(6):629–638, 1976.

[129] George A Hazelrigg. *Fundamentals of Decision Making for Engineering Design and Systems Engineering*. 2012.

[130] HMH Hegge and JC Wortmann. Generic bill-of-material: a new product model. *International Journal of Production Economics*, 23(1):117–128, 1991.

[131] Vera C Hemmelmayr, Karl F Doerner, and Richard F Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195(3):791–802, 2009.

[132] Jade Herbots, Willy Herroelen, and Roel Leus. Dynamic order acceptance and capacity planning on a single bottleneck resource. *Naval Research Logistics (NRL)*, 54 (8):874–889, 2007.

[133] Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling: a survey of recent developments. *Computers & Operations Research*, 25(4):279–302, 1998.

[134] Alain Hertz, Gilbert Laporte, Michel Mittaz, and Kathryn E Stecke. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE transactions*, 30(8):689–694, 1998.

[135] James K Higginson and James H Bookbinder. Policy recommendations for a shipment-consolidation program. *Journal of Business Logistics*, 15(1):87, 1994.

[136] Terry Hill. *Production/operations management: text and cases*. Prentice Hall Hemel Hempstead, 1991.

[137] Lawrence E Holloway, Bruce H Krogh, and Alessandro Giua. A survey of petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems*, 7 (2):151–190, 1997.

[138] Luiz S Homem de Mello and Arthur C Sanderson. And/or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, 6(2):188–199, 1990.

[139] Wallace J Hopp and Mark L Spearman. *Factory physics*. Waveland Press, 2011.

[140] S Jack Hu, J Ko, L Weyand, HA ElMaraghy, TK Lien, Y Koren, Hl Bley, G Chryssolouris, N Nasr, and M Shpitalni. Assembly system design and operations for product variety. *CIRP Annals-Manufacturing Technology*, 60(2):715–733, 2011.

[141] Wei Huang and Lixin Ding. Project-scheduling problem with random time-dependent activity duration times. *IEEE Transactions on Engineering Management*, 58(2):377–387, 2011.

[142] Peter JH Hulshof, Nikky Kortbeek, Richard J Boucherie, Erwin W Hans, and Piet JM Bakker. Taxonomic classification of planning decisions in health care: a structured review of the state of the art in or/ms. *Health Systems*, 1(2):129–175, 2012.

[143] Serguei Iassinovski, Abdelhakim Artiba, and Christophe Fagnart. A generic production rules-based system for on-line simulation, decision making and discrete process control. *International Journal of Production Economics*, 112(1):62–76, 2008.

[144] Michel D Ingham, Robert D Rasmussen, Matthew B Bennett, and Alex C Moncada. Engineering complex embedded systems with state analysis and the mission data system. *Journal of Aerospace Computing, Information, and Communication*, 2(12): 507–536, 2005.

[145] Andrew KS Jardine, Daming Lin, and Dragan Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7):1483–1510, 2006.

[146] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-agent Systems*, 1(1):7–38, 1998.

[147] Jianxin Jiao, Mitchell M Tseng, Qinhai Ma, and Yi Zou. Generic bill-of-materials-and-operations for high-variety production management. *Concurrent Engineering*, 8 (4):297–321, 2000.

[148] Jianxin Roger Jiao, Xiao You, and Arun Kumar. An agent-based framework for collaborative negotiation in the global manufacturing supply chain network. *Robotics and Computer-Integrated Manufacturing*, 22(3):239–255, 2006.

[149] S.M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[150] Ralph L Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press, 1993.

[151] Pinar Keskinocak and Sridhar Tayur. Due date management policies. In *Handbook of Quantitative Supply Chain Analysis*, pages 485–554. Springer, 2004.

[152] Moutaz Khouja. The single-period (news-vendor) problem: literature review and suggestions for future research. *Omega*, 27(5):537–553, 1999.

[153] Moutaz Khouja and Suresh Goyal. A review of the joint replenishment problem literature: 1989–2005. *European Journal of Operational Research*, 186(1):1–16, 2008.

[154] Jinho Kim and KJ Rogers. An object-oriented approach for building a flexible supply chain model. *International Journal of Physical Distribution & Logistics Management*, 35(7):481–502, 2005.

[155] Tamas Kis. Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151(2):307–332, 2003.

[156] Mikhail Yu Kitaev and Vladimir V Rykov. *Controlled queueing systems*. CRC press, 1995.

[157] Thomas Koch, Christoph Krell, and Bernd Kramer. Policy definition language for automated management of distributed systems. In *Proceedings of Second IEEE International Workshop on Systems Management*, pages 55–64. IEEE, 1996.

[158] Tjalling C Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society*, pages 53–76, 1957.

[159] Gyongyi Kovacs and Karen M Spens. Humanitarian logistics in disaster relief operations. *International Journal of Physical Distribution & Logistics Management*, 37(2): 99–114, 2007.

[160] Gy Kovβcs, Karen M Spens, et al. Humanitarian logistics in disaster relief operations. *International Journal of Physical Distribution & Logistics Management*, 37(2):99–114, 2007.

[161] Roelof Kuik, Marc Salomon, and Luk N Van Wassenhove. Batching decisions: structure and models. *European Journal of Operational Research*, 75(2):243–263, 1994.

[162] N Suresh Kumar and R Sridharan. Simulation modelling and analysis of part and tool flow control decisions in a flexible manufacturing system. *Robotics and Computer-Integrated Manufacturing*, 25(4):829–838, 2009.

[163] Andrew Kusiak and Mingyuan Chen. Expert systems for planning and scheduling manufacturing systems. *European Journal of Operational Research*, 34(2):113–130, 1988.

[164] Rob J Kusters and Petra MA Groot. Modelling resource availability in general hospitals design and implementation of a decision support model. *European Journal of Operational Research*, 88(3):428–445, 1996.

[165] H Ahmet Kuyumcu and Ioana Popescu. Deterministic price–inventory management for substitutable products. *Journal of Revenue and Pricing Management*, 4(4):354–366, 2006.

[166] AJD Lambert. Optimal disassembly of complex products. *International Journal of Production Research*, 35(9):2509–2524, 1997.

[167] ID Landau. A survey of model reference adaptive techniques – theory and applications. *Automatica*, 10(4):353–379, 1974.

[168] Henry YK Lau and SO Woo. An agent-based dynamic routing strategy for automated material handling systems. *International Journal of Computer Integrated Manufacturing*, 21(3):269–288, 2008.

[169] Chung-Yee Lee, Sila Çetinkaya, and Wikrom Jaruphongsa. A dynamic model for inventory lot sizing and outbound shipment scheduling at a third-party warehouse. *Operations Research*, 51(5):735–747, 2003.

[170] Doo Yong Lee and Frank DiCesare. Scheduling flexible manufacturing systems using petri nets and heuristic search. *Robotics and Automation, IEEE Transactions on*, 10 (2):123–132, 1994.

[171] Hau L Lee and Christopher S Tang. Modelling the costs and benefits of delayed product differentiation. *Management Science*, 43(1):40–53, 1997.

[172] Severin Lemaignan, Ali Siadat, Jean-Yves Dantan, and Anatoli Semenenko. Mason: A proposal for an ontology of manufacturing domain. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, pages 195–200. IEEE, 2006.

[173] H-K Lin, Jennifer A Harding, and M Shahbaz. Manufacturing system engineering ontology for semantic interoperability across extended project teams. *International Journal of Production Research*, 42(24):5099–5118, 2004.

[174] Shi Qiang Liu and Erhan Kozan. Scheduling trains as a blocking parallel-machine job shop scheduling problem. *Computers & Operations Research*, 36(10):2840–2852, 2009.

[175] Jorge Lobo, R Bhatia, and S Naqvi. Apolicy description language. In *Proceedings of AAAI*, pages 291–298, 1999.

[176] Xinjian Lu and Yigal Gerchak. Minimizing the expected response time of an idled server on a line. *IIE Transactions*, 30(4):401–408, 1998.

[177] Azad M Madni, Weiwen Lin, and Carla C Madni. Ideon$^{\text{TM}}$: An extensible ontology for designing, integrating, and managing collaborative distributed enterprises. *Systems Engineering*, 4(1):35–48, 2001.

[178] James M Magerlein and James B Martin. Surgical demand scheduling: a review. *Health Services Research*, 13(4):418, 1978.

[179] Constantinos Maglaras and Joern Meissner. Dynamic pricing strategies for multi-product revenue management problems. *Manufacturing & Service Operations Management*, 8(2):136–148, 2006.

[180] Avishai Mandelbaum and Alexander L Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized cμ-rule. *Operations Research*, 52(6):836–855, 2004.

[181] Alan S Manne. On the job-shop scheduling problem. *Operations Research*, 8(2): 219–223, 1960.

[182] Francisco P Maturana and Douglas H Norrie. Distributed decision-making using the contract net within a mediator architecture. *Decision Support Systems*, 20(1):53–64, 1997.

[183] Edward J McGavin, Leroy B Schwarz, and James E Ward. Two-interval inventory-allocation policies in a one-warehouse n-identical-retailer distribution system. *Management Science*, 39(9):1092–1107, 1993.

[184] Charles Robert McLean, Y Tina Lee, Guodong Shao, Frank Riddick, and S Leong. *Shop data model and interface specification*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2005.

[185] George H Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[186] Ryszard S Michalski and Gheorghe Tecuci. *Machine learning: A multistrategy approach*. Morgan Kaufmann, 1994.

[187] Hokey Min and Gengui Zhou. Supply chain modeling: past, present and future. *Computers & Industrial Engineering*, 43(1):231–249, 2002.

[188] A Molina and R Bell. A manufacturing model representation of a flexible manufacturing facility. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 213(3):225–246, 1999.

[189] Lars Mönch, John W Fowler, Stéphane Dauzère-Pérès, Scott J Mason, and Oliver Rose. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583–599, 2011.

[190] Lars Mönch, Peter Lendermann, Leon F McGinnis, and Arnd Schirrmann. A survey of challenges in modelling and decision-making for discrete event logistics systems. *Computers in Industry*, 62(6):557–567, 2011.

[191] Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[192] Martin Moser, Dusan P Jokanovic, and Norio Shiratori. An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, 80(3):582–589, 1997.

[193] Wiem Mouelhi-Chibani and Henri Pierreval. Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering*, 58(2):249–256, 2010.

[194] Konstantinos Nikolopoulos, K Metaxiotis, N Lekatis, and Vassilis Assimakopoulos. Integrating industrial maintenance strategy into erp. *Industrial Management & Data Systems*, 103(3):184–191, 2003.

[195] Norman S Nise. *Control Systems Engineering*. John Wiley & Sons, 2007.

[196] OMG OCL. OMG Object Constraint Language (OCL) Version 2.4, 2014. URL `http://www.omg.org/spec/OCL/2.4`.

[197] S Noyan Ogulata, Melik Koyuncu, and Esra Karakas. Personnel and patient scheduling in the high demanded hospital services: a case study in the physiotherapy service. *Journal of Medical Systems*, 32(3):221–228, 2008.

[198] Elias Oliveira and Barbara M Smith. A job-shop scheduling model for the single-track railway scheduling problem. *Research Report Series-University of Leeds School of Computer Studies*, (21), 2000.

[199] Susan Hesse Owen and Mark S Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447, 1998.

[200] Linet Özdamar, Ediz Ekinci, and Beste Küçükyazici. Emergency logistics planning in natural disasters. *Annals of Operations Research*, 129(1-4):217–245, 2004.

[201] Cemal Özgüven, Lale Özbakır, and Yasemin Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6):1539–1548, 2010.

[202] Hervé Panetto, Michele Dassisti, and Angela Tursi. Onto-pdm: Product-driven ontology for product data management interoperability within manufacturing process environment. *Advanced Engineering Informatics*, 26(2):334–348, 2012.

[203] Byung Chun Park. An optimal dwell point policy for automated storage/retrieval systems with uniformly distributed, rectangular racks. *International Journal of Production Research*, 39(7):1469–1480, 2001.

[204] Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian, and Tara Athan. Reaction ruleml 1.0: standardized semantic reaction rules. In *Rules on the Web: Research and Applications*, pages 100–119. Springer, 2012.

[205] Colin Paterson, Gudrun Kiesmüller, Ruud Teunter, and Kevin Glazebrook. Inventory models with lateral transshipments: A review. *European Journal of Operational Research*, 210(2):125–136, 2011.

[206] Pelin Pekgün, Paul M Griffin, and Pinar Keskinocak. Coordination of marketing and production for price and leadtime decisions. *IIE transactions*, 40(1):12–30, 2008.

[207] Charles G Petersen and Gerald Aase. A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92 (1):11–19, 2004.

[208] Dinh-Nguyen Pham and Andreas Klinkert. Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research*, 185(3): 1011–1025, 2008.

[209] Chris N Potts and Mikhail Y Kovalyov. Scheduling with batching: a review. *European Journal of Operational Research*, 120(2):228–249, 2000.

[210] Warren B Powell, Joel A Shapiro, and Hugo P Simao. A representational paradigm for dynamic resource transformation problems. *Annals of Operations Research*, 104 (1):231–279, 2001.

[211] Peter Preston and Erhan Kozan. An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research*, 28(10):983–995, 2001.

[212] OMG PRR. OMG Production Rule Representation (PRR) Version 1.0, 2009. URL `http://www.omg.org/spec/PRR/1.0/`.

[213] Prattana Punnakitikashem, Jay M Rosenberger, and Deborah Buckley Behan. Stochastic programming for nurse assignment. *Computational Optimization and Applications*, 40(3):321–349, 2008.

[214] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons, 2009.

[215] Robin G Qiu and Sanjay B Joshi. A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 29(6):573–586, 1999.

[216] Masoud Rabbani, SMT Fatemi Ghomi, Fariborz Jolai, and NS Lahiji. A new heuristic for resource-constrained project scheduling in stochastic networks using critical chain concept. *European Journal of Operational Research*, 176(2):794–808, 2007.

[217] Gad Rabinowitz, Abraham Mehrez, Ching-Wu Chu, and B Eddy Patuwo. A partial backorder control for continuous review (r,q) inventory system with poisson demand and constant lead time. *Computers & Operations Research*, 22(7):689–700, 1995.

[218] Samuel Raff. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10(2):63–211, 1983.

[219] Abdur Rais and Ana Viana. Operations research in healthcare: a survey. *International Transactions in Operational Research*, 18(1):1–31, 2011.

[220] Peter JG Ramadge and W Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[221] Carmen G Rawls and Mark A Turnquist. Pre-positioning of emergency supplies for disaster response. *Transportation Research Part B: Methodological*, 44(4):521–534, 2010.

[222] Carmen G Rawls and Mark A Turnquist. Pre-positioning and dynamic delivery planning for short-term response following a natural disaster. *Socio-Economic Planning Sciences*, 46(1):46–54, 2012.

[223] Wolfgang Reisig. The expressive power of abstract-state machines. *Computing and Informatics*, 22(3-4):209–219, 2012.

[224] David Riaño, Francis Real, Joan Albert LóPez-Vallverdú, Fabio Campana, Sara Ercolani, Patrizia Mecocci, Roberta Annicchiarico, and Carlo Caltagirone. An ontology-based personalization of health-care knowledge to support clinical decisions for chronically ill patients. *Journal of Biomedical Informatics*, 45(3):429–446, 2012.

[225] Frank Riddick and Y Tina Lee. Representing layout information in the cmsd specification. In *Proceedings of the 2008 Winter Simulation Conference*, pages 1777–1784. IEEE, 2008.

[226] Kees Jan Roodbergen and Iris FA Vis. A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362, 2009.

[227] Robin Roundy, Dietrich Chen, Pan Chen, Metin Çakanyildirim, Michael B Freimer, and Vardges Melkonian. Capacity-driven acceptance of customer orders for a multi-stage batch manufacturing system: models and algorithms. *IIE Transactions*, 37(12): 1093–1105, 2005.

[228] William B Rouse. Engineering perspectives on healthcare delivery: Can we afford technological innovation in healthcare? *Systems Research and Behavioral Science*, 26 (5):573–582, 2009.

[229] Bart Rouwenhorst, B Reuter, V Stockrahm, GJ Van Houtum, RJ Mantel, and WHM Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3):515–533, 2000.

[230] Bernard Roy and B Sussmann. Les problemes dordonnancement avec contraintes disjonctives. *Note D.S.*, 9, 1964.

[231] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[232] Kwangyeol Ryu and Mooyoung Jung. Agent-based fractal architecture and modelling

for developing distributed manufacturing systems. *International Journal of Production Research*, 41(17):4233–4255, 2003.

[233] Norman M Sadeh, David W Hildum, Dag Kjenstad, and Allen Tseng. Mascot: an agent-based architecture for dynamic supply chain creation and coordination in the internet economy. *Production Planning & Control*, 12(3):212–223, 2001.

[234] Said Salhi and Graham K Rand. Incorporating vehicle routing into the vehicle fleet composition problem. *European Journal of Operational Research*, 66(3):313–330, 1993.

[235] Amir Sanayei, S Farid Mousavi, MR Abdi, and Ali Mohaghar. An integrated group decision-making process for supplier selection and order allocation using multi-attribute utility theory and linear programming. *Journal of the Franklin Institute*, 345(7):731–747, 2008.

[236] Tuomas Sandholm and Victor R Lesser. Issues in automated negotiation and electronic commerce: Extending the contract net framework. In *ICMAS*, volume 328, pages 66–73, 1995.

[237] Haralambos Sarimveis, Panagiotis Patrinos, Chris D Tarantilis, and Chris T Kiranoudis. Dynamic modeling and control of supply chain systems: A review. *Computers & Operations Research*, 35(11):3530–3561, 2008.

[238] SC Sarin and CS Chen. The machine loading and tool allocation problem in a flexible manufacturing system. *International Journal of Production Research*, 25(7):1081–1094, 1987.

[239] Shankar Sastry and Marc Bodson. *Adaptive control: stability, convergence and robustness*. Courier Corporation, 2011.

[240] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[241] Katja Schimmelpfeng, Stefan Helber, and Steffen Kasper. Decision support for rehabilitation hospital scheduling. *OR Spectrum*, 34(2):461–489, 2012.

[242] Craig Schlenoff, Michael Gruninger, Florence Tissot, John Valois, Joshua Lubell, and Jintae Lee. *The process specification language (PSL) overview and version 1.0 specification*. Citeseer, 2000.

[243] Jay D Schwartz and Daniel E Rivera. A process control approach to tactical inventory management in production-inventory systems. *International Journal of Production Economics*, 125(1):111–124, 2010.

[244] SCOR. SCOR: The Supply Chain Reference Model Version 11.0, 2012. URL `www.supply-chain.org`.

[245] Willem J Selen and David D Hott. A mixed-integer goal-programming formulation of the standard flow-shop scheduling problem. *Journal of the Operational Research Society*, pages 1121–1128, 1986.

[246] M Kathleen Senehi, Edward J Barkmeyer, Mark E Luce, Steven R Ray, Evan K Wallace, and Sarah Wallace. Manufacturing systems integration initial architecture document. *NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming*, 1991.

[247] Anil Sharma, GS Yadava, and SG Deshmukh. A literature review and future perspectives on maintenance optimization. *Journal of Quality in Maintenance Engineering*, 17(1):5–25, 2011.

[248] Weiming Shen. Distributed manufacturing scheduling using intelligent agents. *IEEE Intelligent Systems*, 17(1):88–94, 2002.

[249] Weiming Shen and Douglas H Norrie. Agent-based systems for intelligent manufacturing: a state-of-the-art survey. *Knowledge and Information Systems*, 1(2):129–156, 1999.

[250] Weiming Shen, Lihui Wang, and Qi Hao. Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *IEEE Transactions on*

Systems, Man, and Cybernetics, Part C: Applications and Reviews, 36(4):563–577, 2006.

[251] Babak Shirazi, Iraj Mahdavi, and Maghsud Solimanpur. Intelligent decision support system for the adaptive control of a flexible manufacturing system with machine and tool flexibility. *International Journal of Production Research*, 50(12):3288–3314, 2012.

[252] Hamid Shojaei, AmirHossein Ghamarian, Twan Basten, Marc Geilen, Sander Stuijk, and Rob Hoes. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management. In *Proceedings of the 46th Annual Design Automation Conference*, pages 917–922. ACM, 2009.

[253] Jean Marcelo Simão and Paulo Cézar Stadzisz. Inference based on notifications: a holonic metamodel applied to control issues. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(1):238–250, 2009.

[254] Jean Marcelo Simão, Cesar Augusto Tacla, and Paulo Cézar Stadzisz. Holonic control metamodel. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 39(5):1126–1139, 2009.

[255] Herbert A Simon. On the application of servomechanism theory in the study of production control. *Econometrica: Journal of the Econometric Society*, pages 247–268, 1952.

[256] Herbert A Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, pages 99–118, 1955.

[257] Morris Sloman. Policy driven management for distributed systems. *Journal of network and Systems Management*, 2(4):333–360, 1994.

[258] Susan A Slotnick. Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, 212(1):1–11, 2011.

[259] Susan A Slotnick and Thomas E Morton. Selecting jobs for a heavily loaded shop with lateness penalties. *Computers & Operations Research*, 23(2):131–140, 1996.

[260] JS Smith, SB Joshi, and RG Qiu. Message-based part state graphs (mpsg): a formal model for shop-floor control implementation. *International Journal of Production Research*, 41(8):1739–1764, 2003.

[261] R Smith. Communication and control in problem solver. *IEEE Transactions on computers*, 29:12, 1980.

[262] R. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29:12, 1980.

[263] Stephen F Smith and Marcel A Becker. An ontology for constructing scheduling systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, pages 120–127, 1997.

[264] Young Jun Son, Richard A Wysk, and Albert T Jones. Simulation-based shop floor control: formal model, model generation and control interface. *IIE Transactions*, 35 (1):29–48, 2003.

[265] Chellappan Sriram and Ali Haghani. An optimization model for aircraft maintenance scheduling and re-assignment. *Transportation Research Part A: Policy and Practice*, 37(1):29–48, 2003.

[266] Graham C Stevens. Integrating the supply chain. *International Journal of Physical Distribution & Logistics Management*, 19(8):3–8, 1989.

[267] James R Stock and Douglas M Lambert. *Strategic logistics management*. McGraw-Hill/Irwin Boston, 2001.

[268] E Szelke and G Markus. A blackboard based multi agent perspective of reactive scheduling. *Artificial Intelligence in Reactive Scheduling*, pages 60–77, 1995.

[269] E Szelke and G Markus. A learning reactive scheduler using cbr/l. *Computers in Industry*, 33(1):31–46, 1997.

[270] Lotfi Tadj and Gautam Choudhury. Optimal design and control of queues. *Top*, 13 (2):359–412, 2005.

[271] Mehrdad Tamiz, Dylan Jones, and Carlos Romero. Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operational Research*, 111(3):569–581, 1998.

[272] Ulrich Aw Tetzlaff. A queueing network model for flexible manufacturing systems with tool management. *IIE Transactions*, 28(4):309–317, 1996.

[273] G. Thiers. *A Model-Based Systems Engineering Methodology to Make Engineering Analysis of Discrete-Event Logistics Systems More Cost-Accessible*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, April 2014.

[274] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[275] Vincent T'kindt and Jean-Charles Billaut. *Multicriteria scheduling: theory, models and algorithms*. Springer Science & Business Media, 2006.

[276] Donald M Topkis. Optimal ordering and rationing policies in a nonstationary dynamic inventory model with n demand classes. *Management Science*, 15(3):160–176, 1968.

[277] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2001.

[278] Evangelos Triantaphyllou. *Multi-criteria decision making methods: a comparative study*, volume 44. Springer Science & Business Media, 2013.

[279] Andy A Tsay, Steven Nahmias, and Narendra Agrawal. Modeling supply chain contracts: A review. In *Quantitative Models for Supply Chain Management*, pages 299–336. Springer, 1999.

[280] Angela Tursi, Hervé Panetto, Gérard Morel, and Michele Dassisti. Ontological approach for products-centric information system interoperability in networked manufacturing enterprises. *Annual Reviews in Control*, 33(2):238–245, 2009.

[281] Karl Ulrich. The role of product architecture in the manufacturing firm. *Research Policy*, 24(3):419–440, 1995.

[282] Mike Uschold, Martin King, Stuart Moralee, and Yannis Zorgios. The enterprise ontology. *The Knowledge Engineering Review*, 13(01):31–89, 1998.

[283] Reha Uzsoy, Chung-Yee Lee, and Louis A Martin-Vega. A review of production planning and scheduling models in the semiconductor industry part i: system characteristics, performance evaluation and production planning. *IIE Transactions*, 24(4): 47–60, 1992.

[284] Kimon P Valavanis. On the hierarchical modeling analysis and simulation of flexible manufacturing systems with extended petri nets. *IEEE Transactions on Systems, Man and Cybernetics*, 20(1):94–110, 1990.

[285] Hendrik Van Brussel, Jo Wyns, Paul Valckenaers, Luc Bongaerts, and Patrick Peeters. Reference architecture for holonic manufacturing systems: Prosa. *Computers in Industry*, 37(3):255–274, 1998.

[286] William Van Melle. A domain-independent production-rule system for consultation programs. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence-Volume 2*, pages 923–925. Morgan Kaufmann Publishers Inc., 1979.

[287] Jan A Van Mieghem. Dynamic scheduling with convex delay costs: The generalized c-mu rule. *The Annals of Applied Probability*, pages 809–833, 1995.

[288] Jan A. Van Mieghem and Nils Rudi. Newsvendor networks: Inventory management and capacity investment with discretionary activities. *Manufacturing & Service Operations Management*, 4(4):313–335, 2002.

[289] Luk N Van Wassenhove and Alfonso J Pedraza Martinez. Using or to adapt supply chain management best practices to humanitarian logistics. *International Transactions in Operational Research*, 19(1-2):307–322, 2012.

[290] Arthur F Veinott Jr. Optimal policy for a multi-product, dynamic, nonstationary inventory problem. *Management Science*, 12(3):206–222, 1965.

[291] Ivan B Vermeulen, Sander M Bohte, Sylvia G Elkhuizen, Han Lameris, Piet JM Bakker, and Han La Poutré. Adaptive resource allocation for efficient patient scheduling. *Artificial Intelligence in Medicine*, 46(1):67–80, 2009.

[292] Guilherme E Vieira, Jeffrey W Herrmann, and Edward Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[293] Jan MH Vissers, Ivo JBF Adan, and Nico P Dellaert. Developing a platform for comparison of hospital admission systems: An illustration. *European Journal of Operational Research*, 180(3):1290–1301, 2007.

[294] Begoña Vitoriano, M Teresa Ortuño, Gregorio Tirado, and Javier Montero. A multicriteria optimization model for humanitarian aid distribution. *Journal of Global Optimization*, 51(2):189–208, 2011.

[295] Birgit Vogel-Heuser, Daniel Witsch, and Uwe Katzke. Automatic code generation from a uml model to iec 61131-3 and system configuration tools. In *International Conference on Control and Automation*, volume 2, pages 1034–1039. IEEE, 2005.

[296] Pavel Vrba, Pavel Tichy, Vladimír Marik, Kenwood H Hall, Raymond J Staron, Francisco P Maturana, and Petr Kadera. Rockwell automation's holonic and multiagent control systems compendium. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 41(1):14–30, 2011.

[297] Harvey M Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[298] Lawrence M Wein. Due-date setting and priority sequencing in a multiclass m/g/1 queue. *Management Science*, 37(7):834–850, 1991.

[299] H Philip Whitaker, Joseph Yamron, and Allen Kezer. *Design of model-reference adaptive control systems for aircraft.* Massachusetts Institute of Technology, Instrumentation Laboratory, 1958.

[300] William W White. Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers. *Networks*, 2(3):211–236, 1972.

[301] D Clay Whybark. Issues in managing disaster relief inventories. *International Journal of Production Economics*, 108(1):228–235, 2007.

[302] DL Woodruff. Subcontracting when there are setups, deadline and tooling costs. In *Proceedings of Intelligent Scheduling Systems Symposium*, pages 337–353, 1992.

[303] Sze-jung Wu, Nagi Gebraeel, Mark Lawley, Yuehwern Yih, et al. A neural network integrated decision support system for condition-based optimal predictive maintenance policy. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(2):226–236, 2007.

[304] Richard A Wysk and Jeffrey S Smith. A formal functional characterization of shop floor control. *Computers & Industrial Engineering*, 28(3):631–643, 1995.

[305] Xiaolong Xue, Xiaodong Li, Qiping Shen, and Yaowu Wang. An agent-based framework for supply chain coordination in construction. *Automation in Construction*, 14 (3):413–430, 2005.

[306] Wei Yi and Linet Özdamar. A dynamic logistics coordination model for evacuation and support in disaster response activities. *European Journal of Operational Research*, 179(3):1177–1193, 2007.

[307] Youngohc Yoon, Tor Guimaraes, and George Swales. Integrating artificial neural networks with rule-based expert systems. *Decision Support Systems*, 11(5):497–507, 1994.

[308] XF Zha, SYE Lim, and SC Fok. Integrated intelligent design and assembly planning: a survey. *The International Journal of Advanced Manufacturing Technology*, 14(9): 664–685, 1998.

[309] Yuyun Zhang, Shaw C Feng, Xiankui Wang, Wensheng Tian, and Ruirong Wu. Object oriented manufacturing resource modelling for adaptive process planning. *International Journal of Production Research*, 37(18):4179–4195, 1999.

[310] Meng Chu Zhou, Frank DiCesare, and Daryl L Rudolph. Design and implementation of a petri net based supervisor for a flexible manufacturing system. *Automatica*, 28 (6):1199–1208, 1992.